E TUTTO DIVENTA PIÙ FACILE!

Programmare in Excel VBA

for dummies®

Estendere Excel con la programmazione VBA

Realizzare nuove applicazioni e componenti

Lavorare meglio con l'automazione VBA

Michael Alexander

John Walkenbach

HOEPLI

Programmare in Excel VBA

for dummies

Programmare in Excel VBA

Michael Alexander
John Walkenbach

EDITORE ULRICO HOEPLI MILANO

Wiley, the Wiley Logo, For Dummies, the Dummies Man Logo, The Dummies Way and related trade dress are trademarks or registered trademarks of John Wiley& Sons, Inc. and/or its affiliates in the United States and/or other countries. Microsoft and Excel are registered trademarks of Microsoft Corporation.
Wiley, il logo Wiley, For Dummies, il logo Dummies Man, Il metodo For Dummies e la relativa grafica sono marchi o marchi registrati di John Wiley& Sons, Inc. e/o dei suoi affiliati negli Stati Uniti e/o in altri paesi. Microsoft ed Excel sono marchi registrati di Microsoft Corporation.
https://marapcana.today - sempre online

Titolo originale: *Excel® VBA Programming For Dummies®*
Copyright © 2019 by John Wiley & Sons, Inc., Hoboken, New Jersey
All rights reserved, including the right of reproduction in whole or in part in any form

Per l'edizione italiana
Copyright © Ulrico Hoepli Editore S.p.A. 2019
via Hoepli 5, 20121 Milano (Italy)
tel. +39 02 864871 – fax +39 02 8052886
e-mail hoepli@hoepli.it

www.hoepli.it

Seguici su Twitter: @Hoepli_1870

https://marapcana.today - sempre online

ISBN EBOOK 978-88-203-9122-5

Traduzione: Paolo Poli
Realizzazione: Maurizio Vedovati – Servizi editoriali (info@iltrio.it)
Copertina: Sara Taglialegne

Realizzazione digitale: Promedia, Torino

A colpo d’occhio https://marapcana.today - sempre online

Introduzione

Parte 1: Introduzione alla programmazione VBA in Excel

CAPITOLO 1: Che cos’è VBA?
CAPITOLO 2: Cerchiamo di capirci qualcosa

Parte 2: Il funzionamento di VBA in Excel

CAPITOLO 3: Usare Visual Basic Editor
CAPITOLO 4: Introduzione al modello a oggetti di Excel
CAPITOLO 5: Routine Sub e Function in VBA
CAPITOLO 6: Uso del registratore di macro di Excel

Parte 3: Concetti di programmazione

CAPITOLO 7: Elementi essenziali del linguaggio VBA
CAPITOLO 8: Gli oggetti Range
CAPITOLO 9: Usare VBA e le funzioni del foglio di lavoro
CAPITOLO 10: Controllare il flusso del programma e prendere decisioni
CAPITOLO 11: Routine ed eventi automatici
CAPITOLO 12: Tecniche di gestione degli errori
CAPITOLO 13: Tecniche di eliminazione dei bug
CAPITOLO 14: Esempi di codice VBA

Parte 4: Comunicare con gli utenti

CAPITOLO 15: Semplici finestre di dialogo
CAPITOLO 16: Concetti di base sulle UserForm
CAPITOLO 17: Utilizzare i controlli per le UserForm
CAPITOLO 18: Tecniche e trucchi per le UserForm
CAPITOLO 19: Richiamare le macro dall’interfaccia utente

Parte 5: Riepilogando...

CAPITOLO 20: Creare nuove funzioni personalizzate per il foglio di lavoro
CAPITOLO 21: Creare componenti aggiuntivi di Excel

Parte 6: La parte dei dieci

CAPITOLO 22: Dieci suggerimenti per l’editor di Visual Basic
CAPITOLO 23: La Guida di VBA
CAPITOLO 24: Dieci cose da fare e da non fare in VBA

Indice analitico

https://marapcana.today - sempre online

Sommario

INTRODUZIONE

Questo libro
Convenzioni utilizzate in questo libro
Controllate le impostazioni di sicurezza
Sciocche premesse
Icone utilizzate
File di esempio online
Come proseguire
L'autore

PARTE 1: INTRODUZIONE ALLA PROGRAMMAZIONE VBA IN EXCEL

CAPITOLO 1: Che cos'è VBA?

Ok, allora che cos'è VBA?
Che cosa potete fare con VBA?
- Inserimento di un testo
- Automatizzare un'attività che svolgete frequentemente
- Automazione di operazioni ripetitive
- Creare un comando personalizzato
- Creare un pulsante personalizzato
- Sviluppare nuove funzioni che operano sul foglio di lavoro
- Creazione di nuovi componenti aggiuntivi per Excel

Vantaggi e svantaggi di VBA
- I vantaggi di VBA
- Gli svantaggi di VBA

VBA in breve
Compatibilità con Excel

CAPITOLO 2: Cerchiamo di capirci qualcosa

Cominciamo dall'inizio
Che cosa faremo ora
Primi passi
Registrazione della macro
Test della macro
Esaminiamo la macro
Modificare la macro
Salvare le cartelle di lavoro contenenti macro
La sicurezza delle macro
Qualcosa di più sulla macro NameAndTime

PARTE 2: IL FUNZIONAMENTO DI VBA IN EXCEL

CAPITOLO 3: Usare Visual Basic Editor

Che cos'è Visual Basic Editor?
- Attivare VBE
- I vari componenti di VBE

Il riquadro Progetto
- Aggiungere un nuovo modulo VBA
- Rimuovere un modulo VBA
- Esportare e importare oggetti

Usare la finestra Codice
- Ridurre a icona e massimizzare le finestre
- Creare un modulo
- Inserire codice VBA in un modulo
- Inserire direttamente il codice
- Usare il registratore di macro
- Copiare il codice VBA

Personalizzare l'ambiente VBA
- Utilizzare la scheda Editor
- Utilizzare la scheda Formato editor
- Usare la scheda Generale
- Usare la scheda Ancoraggio

CAPITOLO 4: Introduzione al modello a oggetti di Excel

Excel è un oggetto?

Muoversi nella gerarchia di oggetti
Le collezioni
Far riferimento agli oggetti
Navigare nella gerarchia
Semplificare i riferimenti agli oggetti
Le proprietà e i metodi
Proprietà di un oggetto
Metodi di un oggetto
Eventi per gli oggetti
Scoprire di più
Utilizzare la Guida di VBA
Usare il Visualizzatore oggetti
Elenco automatico di proprietà e metodi

CAPITOLO 5: Routine Sub e Function in VBA

Routine Sub e Function
Le routine Sub
Le routine Function
Denominare le routine Sub e Function
Eseguire le routine Sub
Eseguire direttamente una routine Sub
Eseguire una routine dalla finestra di dialogo Macro
Eseguire una macro utilizzando un tasto di scelta rapida
Eseguire la routine da un pulsante o da una forma
Eseguire la routine da un'altra routine
Eseguire le routine Function
Richiamare la funzione da una routine Sub
Richiamare una funzione da una formula del foglio di lavoro

CAPITOLO 6: Uso del registratore di macro di Excel

Concetti di base sulla registrazione
Prepararsi alla registrazione
Relativo o assoluto?
Registrare in modalità assoluta
Registrare in modalità relativa
Che cosa viene registrato?
Opzioni di registrazione
Nome macro
Tasto di scelta rapida
L'opzione Memorizza macro in
Descrizione
Questa cosa è efficiente?

PARTE 3: CONCETTI DI PROGRAMMAZIONE

CAPITOLO 7: Elementi essenziali del linguaggio VBA

Utilizzare i commenti nel codice VBA
Variabili, costanti e tipi di dati
Che cosa sono le variabili
Quali sono i tipi di dati di VBA?
Dichiarazione e visibilità delle variabili
Le costanti
Costanti predefinite
Lavorare con le stringhe
Lavorare con le date
Le istruzioni di assegnamento
Esempi di istruzioni di assegnamento
A proposito del segno di uguale
Operatori vari
Lavorare con le matrici: gli array
Dichiarare le matrici
Matrici multidimensionali
Matrici dinamiche
Le etichette

CAPITOLO 8: Gli oggetti Range

Una rapida occhiata
Altri modi per far riferimento a un intervallo
La proprietà Cells
La proprietà Offset
Alcune proprietà utili dell'oggetto Range
La proprietà Value
La proprietà Text
La proprietà Count
Le proprietà Column e Row
La proprietà Address

La proprietà HasFormula
La proprietà Font
La proprietà Interior
La proprietà Formula
La proprietà NumberFormat
Alcuni metodi utili dell'oggetto Range
Il metodo Select
I metodi Copy e Paste
Il metodo Clear
Il metodo Delete

CAPITOLO 9: Usare VBA e le funzioni del foglio di lavoro

Che cos'è una funzione?
Usare le funzioni integrate di VBA
Esempi di funzioni VBA
Funzioni VBA che fanno qualcosa di più che restituire un valore
Conoscere le funzioni VBA
Usare in VBA le funzioni del foglio di lavoro
Esempi di funzioni del foglio di lavoro
Immettere funzioni del foglio di lavoro
Ulteriori informazioni sull'utilizzo delle funzioni del foglio di lavoro
Utilizzare funzioni personalizzate

CAPITOLO 10: Controllare il flusso del programma e prendere decisioni

Seguire il flusso
L'istruzione GoTo
Decisioni, sempre decisioni
La struttura If-Then
La struttura Select Case
I cicli nel codice
I cicli For-Next
Il ciclo Do-While
Il ciclo Do-Until
Usare i cicli For Each-Next con le collezioni

CAPITOLO 11: Routine ed eventi automatici

Preparativi per il grande evento
Gli eventi sono utili?
Programmare routine di gestione degli eventi
Dove va messo il codice VBA?
Scrivere una routine di gestione degli eventi
Esempi introduttivi
L'evento Open per una cartella di lavoro
L'evento BeforeClose per una cartella di lavoro
L'evento BeforeSave per una cartella di lavoro
Esempi di eventi di attivazione
Attivare e disattivare gli eventi in un foglio
Attivare e disattivare gli eventi in una cartella di lavoro
Eventi di attivazione della cartella di lavoro
Altri eventi relativi ai fogli di lavoro
L'evento BeforeDoubleClick
L'evento BeforeRightClick
L'evento Change
Eventi non associati agli oggetti
L'evento OnTime
Eventi per pressioni di tasti

CAPITOLO 12: Tecniche di gestione degli errori

Tipi di errori
Un esempio errato
La macro non è proprio perfetta
La macro non è ancora perfetta
La macro ora è perfetta, vero?
Rinunciare alla perfezione
Gestire gli errori in un altro modo
Revisione della routine EnterSquareRoot
Informazioni sull'istruzione On Error
Gestione degli errori: i dettagli
Recupero dopo un errore
La gestione degli errori in breve
Sapere quando ignorare gli errori
Identificazione di specifici errori
Un errore intenzionale

CAPITOLO 13: Tecniche di eliminazione dei bug

Vari tipi di bug

Identificare i bug
Tecniche di debugging
Esaminare bene il codice
Usare la funzione MsgBox
Inserire istruzioni Debug.Print
Usare il debugger VBA
Il debugger
Impostare i breakpoint nel codice
Usare la finestra Immediata
Attraversare il codice, passo dopo passo
Usare la finestra Espressioni di controllo
Usare la finestra Variabili locali
Suggerimenti per la riduzione dei bug

CAPITOLO 14: Esempi di codice VBA

Lavorare con gli intervalli
Copiare un intervallo
Copiare un intervallo di dimensioni variabili
Selezionare fino alla fine una riga o una colonna
Selezionare una riga o una colonna
Spostare un intervallo
Eseguire un ciclo su un intervallo in modo efficiente
Eseguire un ciclo su un intervallo in modo efficiente (Parte II)
Richiedere un valore per una cella
Determinare il tipo della selezione
Identificare una selezione multipla
Modificare le impostazioni di Excel
Modificare le impostazioni booleane
Modificare le impostazioni non booleane
Lavorare sui grafici
AddChart vs AddChart2
Cambiare tipo di grafico
Cicli nella collezione ChartObjects
Modificare le proprietà di un grafico
Applicare formattazioni al grafico
Suggerimenti per velocizzare VBA
Disattivare l'aggiornamento dello schermo
Disattivare il calcolo automatico
Eliminare i fastidiosi messaggi di allarme
Semplificare i riferimenti agli oggetti
Dichiarare i tipi di variabile
Usare la struttura With-End With

PARTE 4: COMUNICARE CON GLI UTENTI

CAPITOLO 15: Semplici finestre di dialogo

Alternative alle UserForm
La funzione MsgBox
Visualizzare una semplice finestra
Ottenere una risposta da una finestra MsgBox
Personalizzare le finestre MsgBox
La funzione InputBox
La sintassi di InputBox
Un esempio di InputBox
Un altro tipo di InputBox
Il metodo GetOpenFilename
La sintassi del metodo GetOpenFilename
Un esempio d'uso di GetOpenFilename
Il metodo GetSaveAsFilename
Ottenere un nome di cartella
Visualizzare le finestre di dialogo di Excel

CAPITOLO 16: Concetti di base sulle UserForm

Quando utilizzare una UserForm
Creare una UserForm: una panoramica
Usare le UserForm
Inserire una nuova UserForm
Aggiungere controlli a una UserForm
Modificare le proprietà di un controllo della UserForm
Visualizzare la finestra Codice della UserForm
Visualizzare una UserForm
Utilizzare le informazioni di una UserForm
Una UserForm di esempio
Creare la UserForm
Aggiungere i CommandButton
Aggiungere gli OptionButton
Aggiungere la routine di gestione degli eventi

Creare la macro per visualizzare la finestra di dialogo
Rendere disponibile la macro
Provare la macro

CAPITOLO 17: Utilizzare i controlli per le UserForm

I controlli per le finestre di dialogo
Aggiungere controlli
Le proprietà dei controlli
I controlli della finestra di dialogo: tutti i dettagli
Il controllo CheckBox (Casella di controllo)
Il controllo ComboBox (Casella combinata)
Il controllo CommandButton (Pulsante di comando)
Il controllo Frame (Cornice)
Il controllo Image (Immagine)
Il controllo Label (Etichetta)
Il controllo ListBox (Casella di riepilogo)
Il controllo MultiPage (Pagine)
Il controllo OptionButton (Pulsante di opzione)
Il controllo RefEdit
Il controllo ScrollBar (Barra di scorrimento)
Il controllo SpinButton (Pulsante di selezione)
Il controllo TabStrip (Schede)
Il controllo TextBox (Casella di testo)
Il controllo ToggleButton (Pulsante interruttore)
Gestire i controlli nella finestra di dialogo
Spostare e ridimensionare i controlli
Allineare e distribuire i controlli
Gestire l'input da tastiera
Impostare i tasti di scelta rapida
Provare una UserForm
L'estetica delle finestre di dialogo

CAPITOLO 18: Tecniche e trucchi per le UserForm

Usare le finestre di dialogo
Un esempio di UserForm
Creare la finestra di dialogo
Scrivere il codice per visualizzare la finestra di dialogo
Rendere disponibile la macro
Provare la finestra di dialogo
Aggiungere le routine di gestione degli eventi
Convalidare i dati
Ora la finestra di dialogo funziona
Un esempio di ListBox
Compilare il contenuto di una ListBox
Determinare l'elemento selezionato
Determinare le selezioni multiple
Selezionare un intervallo
Utilizzare più insiemi di OptionButton
Utilizzare uno SpinButton e una TextBox
Utilizzare una UserForm come indicatore di avanzamento
Creare la finestra di dialogo dell'indicatore di avanzamento
Le routine
Come funziona questo esempio
Creare una finestra di dialogo non modale a schede
Visualizzare un grafico in una UserForm
Una checklist per le finestre di dialogo

CAPITOLO 19: Richiamare le macro dall'interfaccia utente

Personalizzare la barra multifunzione
Personalizzare manualmente la barra multifunzione
Aggiungere una macro alla barra multifunzione
Personalizzare la barra multifunzione con il linguaggio XML
Personalizzare i menu contestuali
Aggiungere un nuovo elemento al menu contestuale della cella
Che cosa cambia rispetto a Excel 2007?

PARTE 5: RIEPILOGANDO...

CAPITOLO 20: Creare nuove funzioni personalizzate per il foglio di lavoro

Perché creare funzioni personalizzate?
Concetti di base sulle funzioni VBA
Scrivere nuove funzioni
Gli argomenti delle funzioni
Una funzione senza argomenti
Una funzione con un argomento
Una funzione con due argomenti

Una funzione con un argomento intervallo
Una funzione con un argomento opzionale
Introduzione alle funzioni wrapper
La funzione NumberFormat
La funzione ExtractElement
La funzione SayIt
La funzione IsLike
Funzioni che restituiscono una matrice
Restituire una matrice dei nomi dei mesi
Restituire una lista ordinata
La finestra di dialogo Inserisci funzione
Visualizzare la descrizione della funzione
Aggiungere le descrizioni degli argomenti
Descrizioni degli argomenti delle funzioni

CAPITOLO 21: Creare componenti aggiuntivi di Excel
D'accordo… ma che cos'è un componente aggiuntivo?
Perché creare dei componenti aggiuntivi?
Usare i componenti aggiuntivi
Concetti di base sui componenti aggiuntivi
Un esempio di componente aggiuntivo
Impostare la cartella di lavoro
Provare la cartella di lavoro
Aggiungere informazioni descrittive
Proteggere il codice VBA
Creare il componente aggiuntivo
Aprire il componente aggiuntivo
Distribuire il componente aggiuntivo
Modificare il componente aggiuntivo

PARTE 6: LA PARTE DEI DIECI

CAPITOLO 22: Dieci suggerimenti per l'editor di Visual Basic
Inserire blocchi di commenti
Copiare più righe di codice contemporaneamente
Spostarsi tra moduli e routine
Teletrasportare le funzioni
Rimanere nella routine corrente
Esecuzione passo-passo del codice
Passare a una determinata riga del codice
Arrestare il codice in un punto predefinito
Vedere per intero il valore delle variabili
Disattivare il controllo automatico della sintassi

CAPITOLO 23: La Guida di VBA
Lasciare che Excel scriva il codice per voi
Trarre codice da Internet
Sfruttare i forum degli utenti
Visitare blog di esperti
Video su YouTube
Partecipare in diretta e online ai corsi di formazione
Apprendere dal Microsoft Office Dev Center
Sezionare i file di Excel disponibili in azienda
Chiedere al guru locale di Excel

CAPITOLO 24: Dieci cose da fare e da non fare in VBA
Dichiarate tutte le variabili
Non confondete password con sicurezza
Pulite il codice
Non mettete tutto in una sola routine
Prendete in considerazione altri software
Non date per scontata l'abilitazione delle macro
Prendete l'abitudine di sperimentare
Non supponete che il vostro codice funzioni con altre versioni di Excel
Fate attenzione ai vostri utenti
Non dimenticate i backup

INDICE ANALITICO

Informazioni sul Libro

Circa gli autori

Introduzione

Benvenuti futuri programmatori di Excel…

Senza dubbio avrete le vostre ragioni per aver acquistato un libro sulla programmazione VBA. Magari avete un nuovo lavoro (congratulazioni!). Magari state cercando di automatizzare delle attività ripetitive di elaborazione dei dati. O forse siete solo *nerd*. Qualunque sia la ragione, innanzitutto grazie per aver scelto questo libro.

In questo libro troverete tutto ciò di cui avete bisogno per iniziare subito a lavorare in VBA. E anche se non avete la più pallida idea di cosa sia la programmazione, questo libro può aiutarvi. A differenza della maggior parte dei libri di programmazione, è ricco di informazioni progettate per includere tutto ciò che è necessario sapere per apprendere rapidamente le basi della programmazione VBA.

Questo libro

Recandovi in una qualsiasi grande libreria (di persona, ma anche online) troverete molti libri su Excel. Una rapida panoramica potrà aiutarvi a decidere se questo libro è davvero utile per voi. Questo libro…

- È pensato per gli utenti di Excel di livello intermedio/avanzato che desiderano aggiornarsi nella programmazione VBA (Visual Basic for Applications).
- Non richiede alcuna precedente esperienza di programmazione.
- Presenta tutti i comandi più comunemente usati.
- È adatto a Excel 2013, 2016 e 2019.
- Vuole anche farvi sorridere, di tanto in tanto.

Se state utilizzando Excel 2003, dovrete procurarvi un libro molto più datato (e vi meritate anche un abbraccio). Se state utilizzando Excel 2007 o 2010, questo libro *potrebbe* anche andare bene alle vostre esigenze, ma alcune cose sono cambiate. Probabilmente sarebbe più adatta un'edizione meno aggiornata.

Ebbene sì - questo *non è* un libro introduttivo su Excel. Se state cercando un testo generico, consultate uno dei seguenti libri, entrambi pubblicati da Hoepli:

- *Excel 2019 For Dummies* di Greg Harvey
- *Excel 2019 Formule e funzioni* di Paolo Poli

Questi libri sono disponibili anche per le versioni precedenti di Excel.

Notate che il titolo di questo libro non è "*Guida completa alla programmazione VBA in Excel for dummies*". Questo libro non copre tutti gli aspetti della programmazione in Excel, ma, d'altra parte, probabilmente non volete sapere proprio tutto su questo argomento.

Se leggendo questo libro scoprite di essere interessati a un libro di programmazione Excel più completo, potreste provare *Microsoft Excel 2019 Power Programming with VBA*, pubblicato da John Wiley. E anche in questo caso, sono disponibili anche edizioni per le versioni precedenti di Excel.

Convenzioni utilizzate in questo libro

Tutti i libri per computer comprendono un paragrafo come questo (sembra quasi un obbligo di legge). Decidete voi: leggetelo o saltatelo.

A volte nomino delle combinazioni di tasti, questo significa che dovete tener premuto un tasto e poi premerne un altro. Per esempio, Ctrl + Z significa che dovete tenere premuto il tasto Ctrl mentre premete Z.

Per i comandi di menu, uso un particolare simbolo per separare gli elementi della barra multifunzione o del menu. Per esempio, il seguente comando crea un intervallo con nome in un foglio di lavoro:

```
Formule ➪ Nomi definiti ➪ Definisci nome
```

Formule è la scheda nella parte superiore della barra multifunzione; Nomi definiti è il gruppo di comandi della barra multifunzione; Definisci nome è il comando da richiamare.

Visual Basic Editor utilizza ancora il classico sistema a menu e barre di strumenti. Quindi Strumenti ➪ Opzioni significa scegliere il menu Strumenti e poi scegliere la voce di menu Opzioni.

La programmazione Excel implica lo sviluppo di codice, ovvero istruzioni che poi Excel dovrà eseguire.

Tutto il codice del libro è scritto in font monospaziato:

```
Range("A1:A12").Select
```

Alcune righe di codice sono talmente lunghe da non rientrare nei margini del libro stampato. In questi casi, utilizzo la sequenza di caratteri di continuazione di riga standard in VBA: uno spazio seguito da un carattere di sottolineatura. Ecco un esempio:

```
Selection.PasteSpecial Paste:=xlValues, _
Operation:=xlNone, SkipBlanks:=False, _
Transpose:=False
```

Quando inserite questo codice, potete scriverlo così come è o su un'unica riga (omettendo ovviamente la combinazione spazio + sottolineatura).

Controllate le impostazioni di sicurezza

Là fuori il mondo è crudele. Può sempre esserci qualche truffatore che cerca di approfittarsi di voi e di causarvi qualche problema. Il mondo dell'informatica è altrettanto crudele. Probabilmente siete a conoscenza di virus che possono causare brutte sorprese al vostro sistema. Ma i virus possono essere inseriti anche in un file di Excel? Sì, è così. In effetti, è relativamente facile scrivere un virus usando il linguaggio VBA. E così un utente inconsapevole potrebbe aprire un file Excel e diffondere il virus ad altre cartelle di lavoro di Excel e ad altri sistemi.

Nel corso degli anni, Microsoft si è sempre preoccupata dei problemi relativi alla sicurezza. Questa è una buona cosa, ma anche che gli utenti di Excel devono comprendere il funzionamento delle cose. Potete controllare le impostazioni di sicurezza di Excel selezionando File ➪ Opzioni ➪ Centro protezione ➪ Impostazioni Centro protezione. Troverete una miriade di opzioni, e si dice anche che chi ha aperto quella finestra di dialogo non è più tornato indietro.

Se fate clic sulla scheda Impostazioni delle macro (sul lato sinistro della finestra di dialogo Centro protezione), troverete le seguenti opzioni:

- **Disabilita tutte le macro senza notifica.** Le macro non funzioneranno, neppure a pregare in cinese.
- **Disabilita tutte le macro con notifica.** Quando aprite una cartella di lavoro contenente macro, compare la barra dei messaggi contenente un'opzione, con la quale potete abilitare le macro; oppure (se è aperta la finestra di Visual Basic Editor) viene visualizzato un messaggio che chiede se desiderate abilitare le macro.
- **Disabilita tutte le macro tranne quelle con firma digitale.** Potrete eseguire solo le macro con firma digitale (ma se la loro firma non è considerata attendibile, viene comunque visualizzato l'avviso di sicurezza).
- **Abilita tutte le macro.** Tutte le macro vengono abilitate, senza avvisi. Questa opzione è sconsigliata poiché permette di eseguire codice potenzialmente pericoloso.

Considerate la seguente situazione: trascorrete un'intera settimana a scrivere e perfezionare un magnifico programma VBA che rivoluzionerà il lavoro in azienda. Lo provate a fondo e lo inviate al capo. Lui vi chiama in ufficio e afferma che la vostra bella macro non funziona. Che cosa sta succedendo? Probabilmente le impostazioni di sicurezza del computer del capo non consentono l'esecuzione di macro. O magari ha scelto di seguire il suggerimento di Microsoft e ha disabilitato tutte le macro quando ha aperto il file.

La lezione? Solo perché una cartella di lavoro di Excel contiene una macro, questo non garantisce che la macro verrà mai eseguita. Tutto dipende dalle impostazioni di sicurezza e dal fatto che l'utente sceglierà di abilitare o disabilitare le macro per quel file.

Con questo libro, è necessario abilitare le macro per i file con i quali si lavora. Il mio consiglio è quello di impostare il secondo livello di sicurezza. Quindi, quando aprite un file che avete creato, potete semplicemente abilitare le macro. Se aprite un file di qualcuno che non conoscete, dovreste disabilitare le macro e controllare il codice VBA per assicurarvi che non contenga nulla di distruttivo o pericoloso. Di solito, è abbastanza facile identificare il codice VBA sospetto.

Un'altra opzione consiste nel designare una cartella attendibile. Scegliete File ➪ Opzioni ➪ Centro protezione ➪ Impostazioni Centro protezione. Selezionate l'opzione Posizioni attendibili e indicate una determinata cartella come posizione attendibile. Memorizzate le vostre cartelle di lavoro ed Excel non vi infastidirà più con i suoi dubbi sull'abilitazione delle macro. Per esempio, se scaricate i file degli esempi di questo libro, potete inserirli in una posizione attendibile.

Sciocche premesse

Chi scrive libri, di solito ha in mente un lettore di riferimento. Il seguente elenco descrive, più o meno, l'ipotetico lettore di questo libro.

- Avete accesso a un PC al lavoro e probabilmente anche a casa. E questi computer sono collegati a Internet.
- State usando Excel 2013, 2016 o 2019.
- State usando i computer già da diversi anni.
- Usate Excel frequentemente al lavoro e ritenete di essere più esperti di Excel rispetto all'impiegato medio.
- Dovete fare in modo che Excel esegua alcune cose che al momento non riuscite a ottenere.
- Avete poca o nessuna esperienza di programmazione.
- Sapete che la Guida di Excel può effettivamente essere utile. Usatela; questo libro non tratta tutti gli argomenti. Se specificate nella Guida dei termini validi, sarete in grado di colmare tutti gli elementi mancanti.
- Dovete eseguire alcuni compiti specifici, ma avete una limitata tolleranza per i noiosi "mattoni" dedicati ai computer.

Icone utilizzate

CONSIGLIO

Non saltate le informazioni contrassegnate da questa icona. Identifica una scorciatoia che può farvi risparmiare tempo (e permettervi anche di lasciare l'ufficio a un orario ragionevole). Questa icona vi fa sapere anche che il codice in questione è disponibile sul Web. Scaricatelo per evitare di digitarlo.

RICORDA!

Questa icona evidenzia le informazioni che dovreste tenere a mente, perché potranno rivelarsi utili in futuro.

PER I PIÙ CURIOSI

Questa icona presenta argomenti più approfonditi, che potete benissimo saltare se non vi interessa scoprire qualcosa di più.

WARNING

Questa icona presenta informazioni che dovreste davvero considerare, per evitare situazioni molto spiacevoli.

File di esempio online

Questo libro ha un proprio sito web dove potete scaricare i file di esempio. Per ottenere questi file potete accedere al seguente indirizzo web.

```
www.hoepliedi tore.it/9083-9
```

Il fatto di poter usare i file di esempio vi farà risparmiare molto tempo di digitazione. Meglio ancora, potrete provarli e sperimentare varie modifiche. In effetti, la sperimentazione è il modo migliore per imparare a usare il linguaggio VBA.

Come proseguire

Questo libro contiene tutto il necessario per imparare a programmare in VBA a un livello avanzato. Il libro parte dalle basi della registrazione di macro e sviluppa l'argomento capitolo per capitolo.

Se siete assolutamente neofiti nel campo delle macro di Excel, iniziate dalla Parte 1 per scoprire i fondamenti della registrazione di macro. Se avete già una certa esperienza, ma volete conoscere meglio il linguaggio VBA, leggete la Parte 2. Qui troverete una descrizione sintetica del funzionamento di VBA, oltre alle basi necessarie per implementare il codice.

Infine, se avete familiarità con i concetti di programmazione e volete solo scoprire le tecniche più avanzate, come la creazione di funzioni personalizzate e componenti aggiuntivi, saltate direttamente alla Parte 4. Esiste anche un piccolo promemoria di scorciatoie utili. Per trovarlo, andate su www.hoepliedi tore.it/9083-9 e cercate la "Tabella del programmatore Excel VBA".

L'autore

Michael Alexander è Microsoft Certified Application Developer (MCAD) e autore di numerosi libri sull'analisi aziendale con Microsoft Access e Microsoft Excel. Ha oltre 20 anni di esperienza nella consulenza e nello sviluppo di soluzioni con Microsoft Office. Mike è stato nominato Microsoft MVP per i suoi continui contributi alla comunità di Excel. Potete trovare Mike su www.datapigtechnologies.com.

1

Introduzione alla programmazione VBA in Excel

IN QUESTA PARTE...

- Scoprire Visual Basic for Applications.
- Alcuni esempi di cosa si può fare con VBA.
- Un esempio di una classica sessione di programmazione in Excel.
- Scoprire come Excel si occupa della sicurezza delle macro.

Capitolo 1

Che cos’è VBA?

IN QUESTO CAPITOLO

- **Una panoramica concettuale su VBA**
- **Scoprire cosa si può fare con VBA**
- **Scoprire i vantaggi e gli svantaggi dell’utilizzo di VBA**
- **Scoprire i punti deboli di VBA**
- **Restare compatibili con Excel**

Se siete impazienti di mettere le mani sulla programmazione VBA, portate un po’ di pazienza. Questo capitolo è praticamente privo di materiale didattico pratico. Tuttavia, contiene alcune informazioni di base che vi aiuteranno a diventare programmatori Excel. In altre parole, questo capitolo apre la strada a tutto ciò che segue e dà un’idea di come la programmazione di Excel si inserisca nello schema generale. Ma non è così noioso come potreste immaginare, quindi provate a resistere all’impulso di saltare direttamente al Capitolo 2.

Ok, allora che cos’è VBA?

VBA, abbreviazione di *Visual Basic for Applications*, è un linguaggio di programmazione sviluppato da Microsoft (sì, quella società che cerca di farvi comprare una nuova versione di Windows ogni tot anni). Excel, insieme agli altri prodotti del pacchetto Microsoft Office, include (senza costi aggiuntivi) il linguaggio VBA. In poche parole, VBA è lo strumento utilizzato per sviluppare programmi che controllano Excel.

Immaginate un robot intelligente che conosca tutto di Excel. Questo robot è in grado di leggere le istruzioni e può anche utilizzare Excel in modo rapido e preciso. Quando volete che il robot esegua qualcosa in Excel, scrivete una serie di istruzioni usando alcuni codici speciali. Poi chiedete a una specie di robot di seguire le vostre istruzioni, mentre vi sedete a bere una bella bibita. Questo è VBA: un linguaggio di codici per robot. Ah… comunque Excel viene fornito *con* VBA, ma *senza* robot e bibita.

PER I PIÙ CURIOSI

QUALCHE PAROLA SULLA TERMINOLOGIA

La terminologia usata nella programmazione di Excel può essere un po’ contorta. Per esempio, VBA è un linguaggio di programmazione, ma funge anche da linguaggio per macro. In questo contesto, una macro è un insieme di istruzioni che Excel esegue per imitare le pressioni di tasti e le azioni del mouse. Come si chiama un qualcosa scritto in VBA ed eseguito da Excel? È una *macro* o è un *programma*? La Guida di Excel fa spesso riferimento alle procedure VBA chiamandole *macro*, pertanto in questo libro utilizzerò questa stessa terminologia. Ma se le chiamerete *programmi* non vi mangerà nessuno.

In questo libro vedrete anche utilizzare il termine *automatizzare*. Ciò indica che una serie di passaggi manuali vengono completati automaticamente. Se, per esempio, scrivete una macro che applica un colore ad alcune celle, poi stampa il foglio di lavoro e infine rimuove il colore, avrete *automatizzato* questi tre passaggi.

A proposito, *macro* non significa “Mettiamo Assieme Cose alla Rinfusa Ovunque”. La parola deriva dal greco *μακρός* (*makros*), che significa grande. Fra le altre cose parla del vostro stipendio, dopo essere diventati esperti nella programmazione di macro.

Che cosa potete fare con VBA?

Probabilmente sapete che Excel viene impiegato per una miriade di attività, tutte differenti. Ecco alcuni esempi.

- Analizzare dati scientifici.
- Preparare budget e previsioni.
- Creare fatture e altri moduli.
- Sviluppare grafici partendo dai dati.
- Gestire elenchi (di nomi di clienti, voti di studenti o idee regalo per le feste… a proposito, una bella crostata alla frutta sarebbe molto gradita).
- Questo, quello e anche quell'altro.

L'elenco potrebbe continuare all'infinito, ma vi siete fatti un'idea. Il punto è semplicemente che Excel viene utilizzato per un'ampia gamma di attività, e chiunque stia leggendo questo libro ha sicuramente esigenze e aspettative diverse a proposito di Excel. Una cosa che praticamente tutti i lettori hanno in comune è la *necessità di automatizzare alcuni aspetti di Excel*. Quello, caro lettore, è proprio "pane per i denti" di VBA.

Per esempio, potete creare un programma VBA per importare alcuni valori e poi formattare e stampare il report delle vendite del mese. Dopo aver sviluppato e sottoposto a test il programma, potrete eseguire la macro con un singolo comando, facendo in modo che Excel esegua automaticamente molte procedure potenzialmente onerose. Invece di ripetere una noiosa sequenza di comandi, potete fare clic su un pulsante, dare un'occhiata a Facebook e ingannare un po' il tempo mentre la macro lavora al posto vostro.

I paragrafi seguenti descrivono brevemente alcuni usi comuni delle macro VBA. Una o due di queste potrebbero darvi qualche idea.

Inserimento di un testo

Se vi capita spesso di inserire nei fogli di lavoro il nome, l'indirizzo e il numero di telefono dell'azienda, potete creare una macro che faccia questo lavoro per voi. Potete anche estendere questo concetto a piacimento. Per esempio, potreste sviluppare una macro che generi automaticamente l'elenco di tutti i venditori dell'azienda.

Automatizzare un'attività che svolgete frequentemente

Siete il direttore delle vendite e dovete stilare un report sulle vendite del mese per accontentare il capo. Se l'attività è semplice, potete sviluppare un programma VBA che lo generi per voi. Il vostro capo rimarrà colpito dalla qualità costantemente elevata dei vostri report e vi promuoverà: avrete un nuovo lavoro per il quale non siete per nulla qualificati.

Automazione di operazioni ripetitive

Se vi capita di dover eseguire la stessa azione su, diciamo, dodici diverse cartelle di lavoro di Excel, potete registrare una macro mentre svolgete l'attività sulla prima cartella e poi fare in modo che la macro ripeta la propria azione su tutte le altre cartelle di lavoro. La cosa bella è che Excel non si annoia, mai. Il registratore di macro di Excel è simile al tasto REC della videocamera. Ma senza videocamera e senza il problema delle batterie scariche.

Creare un comando personalizzato

Vi trovate a eseguire spesso sempre la stessa sequenza di comandi di Excel? Dedicate una manciata di secondi a sviluppare una macro che combina tutti questi comandi in un unico comando personalizzato, che potete eseguire con una combinazione di tasti o un clic. Magari non risparmierete poi così tanto tempo, ma probabilmente l'operazione sarà più precisa. E il tipo che vi osserva seduto accanto a voi in ufficio resterà a bocca aperta.

Creare un pulsante personalizzato

Potete personalizzare la barra di accesso rapido con nuovi vostri pulsanti che lanciano le vostre macro. I colleghi sono sempre impressionati dai pulsanti che svolgono una qualche *magia*. Ma se volete *davvero* stupire i vostri colleghi, potete perfino aggiungere nuovi pulsanti alla barra multifunzione.

Sviluppare nuove funzioni che operano sul foglio di lavoro

Sebbene Excel includa centinaia di funzioni integrate (come `SOMMA` e `MEDIA`), potete anche creare nuove funzioni personalizzate in grado di semplificare notevolmente le formule. Sarete sorpresi di quanto sia facile crearle (ne parliamo nel Capitolo 20). Ma c'è di più: la finestra di dialogo Inserisci funzione elencherà anche le vostre nuove funzioni, insieme a quelle "di serie". Tanta roba davvero!

Creazione di nuovi componenti aggiuntivi per Excel

Probabilmente avrete una certa familiarità con alcuni dei componenti aggiuntivi forniti con Excel. Per esempio, un componente aggiuntivo molto popolare è Strumenti di analisi. Potete utilizzare VBA per sviluppare i vostri componenti aggiuntivi speciali.

Vantaggi e svantaggi di VBA

Questo paragrafo elenca gli aspetti positivi di VBA e parla anche del suo lato oscuro.

I vantaggi di VBA

Potete automatizzare quasi tutto in Excel. Per farlo, basta scrivere le istruzioni da far eseguire da Excel. L'automazione di un'attività tramite VBA offre diversi vantaggi.

- Excel eseguirà l'attività sempre allo stesso modo. In molti casi, la coerenza è una gran bella cosa.
- Excel eseguirà l'attività molto più velocemente di quanto possiate mai farlo voi (a meno che, naturalmente, non siate Clark Kent).
- Se programmate una macro, Excel eseguirà l'attività sempre senza commettere errori (il che probabilmente non si può dire di voi, indipendentemente dall'attenzione che ci mettete).
- Se impostate correttamente le cose, la vostra macro potrà essere utilizzata da chiunque, anche se non conosce Excel.
- Potrete inserire in Excel operazioni altrimenti impossibili, cosa che potrebbe rendervi molto popolari in ufficio.
- Per ripetere le attività lunghe e noiose, non sarete costretti a sedervi al computer a osservare lo schermo. Potrete chiedere a Excel di svolgere il lavoro al posto vostro, mentre andate in gita alla macchinetta del caffè.

Gli svantaggi di VBA

È giusto dare altrettanto spazio anche agli svantaggi (o ai *potenziali* svantaggi) di VBA.

- Dovete imparare a scrivere programmi VBA (ma non è proprio questo lo scopo per cui avete comprato questo libro?). Fortunatamente, la cosa non è così difficile come potete pensare.
- Tutti coloro che dovranno utilizzare i vostri programmi VBA dovranno anche avere una propria copia di Excel. Sarebbe bello se fosse possibile premere un pulsante che trasforma la vostra macro Excel/VBA in un programma a sé stante, ma non è ancora possibile (e probabilmente non lo sarà mai).
- A volte, le cose si mettono male. In altre parole, non potete pensare che il vostro programma VBA funzionerà sempre correttamente in tutte le circostanze. Benvenuti nel mondo del *debug* e, nel caso in cui altri stiano usando le vostre macro, dovrete fare anche da supporto tecnico.
- VBA è un bersaglio mobile. Come sapete, Microsoft aggiorna continuamente Excel. Anche se Microsoft si impegna moltissimo a mantenere la compatibilità tra le versioni, potreste scoprire che il codice VBA che avete scritto non funziona correttamente con le versioni troppo datate di Excel o con una versione futura.

VBA in breve

Tanto per introdurre l'argomento, ecco un rapido e sintetico riassunto di ciò che fa VBA.

- **Eseguite azioni in VBA scrivendo (o registrando) del codice in un modulo VBA.** Potete visualizzare e modificare i moduli VBA utilizzando Visual Basic Editor (VBE).
- **Un modulo VBA è costituito da procedure** `Sub`. Una procedura `Sub` non ha nulla a che fare con i sommergibili o con pinne e bombole. È piuttosto una sezione di codice che esegue un'azione su determinati oggetti (un po' di pazienza e ne parliamo). L'esempio seguente mostra una semplice procedura `Sub` chiamata `AddEmUp`. Questo eccezionale programma, quando viene eseguito, mostra il risultato della somma di 1 più 1:

  ```
  Sub AddEmUp()
       Sum = 1 + 1
       MsgBox "The answer is " & Sum
  End Sub
  ```

  Una procedura `Sub` che non funziona correttamente è detta `Sub` standard.
- **Un modulo VBA può anche avere procedure** `Function`**.** Una procedura `Function` restituisce un singolo valore. Potete richiamarla da un'altra procedura VBA o anche usarla direttamente in una formula del foglio di lavoro. Ecco un esempio di procedura `Function` (chiamata `AddTwo`). La funzione accetta due numeri (chiamati *argomenti*) e ne restituisce la somma:

  ```
  Function AddTwo(arg1, arg2)
       AddTwo = arg1 + arg2
  ```

```
End Function
```

Una procedura `Function` che non funziona correttamente è detta disfunzionale.

» **VBA manipola degli oggetti.** Excel fornisce decine e decine di oggetti che possono essere soggetti a una manipolazione. Tra gli esempi di oggetti vi sono la cartella di lavoro, un foglio di lavoro, un intervallo di celle, un grafico e una forma grafica. Avete davvero molti oggetti a vostra disposizione e potete manipolarli tutti usando codice VBA.

» **Gli oggetti sono disposti in una gerarchia.** Gli oggetti possono fungere da *contenitori* per altri oggetti. La parte superiore della gerarchia di oggetti è Excel. Excel stesso è un oggetto chiamato `Application`. L'oggetto `Application` contiene altri oggetti, come oggetti `Workbook` e oggetti `Add-In`. L'oggetto `Workbook` può contenere altri oggetti, come gli oggetti `Worksheet` e `Chart`. Un oggetto `Worksheet` può contenere oggetti come `Range` e `PivotTable`. Il termine *modello di oggetto* fa riferimento alla disposizione di questi oggetti. Per ulteriori informazioni consultate il Capitolo 4.

» **Gli oggetti dello stesso tipo formano una collezione.** La collezione `Worksheets`, per esempio, è composta da tutti i fogli di lavoro di una determinata cartella di lavoro. La raccolta `Charts` è composta da tutti gli oggetti `Chart` di una cartella di lavoro. Le collezioni sono esse stesse oggetti.

» **Per far riferimento a un oggetto se ne specifica la posizione nella gerarchia di oggetti, utilizzando come separatore un punto.** Per esempio, potete fare riferimento alla cartella di lavoro `Cartel1.xlsx` come:

```
Application.Workbooks("Cartel1.xlsx")
```

Questo si riferisce alla cartella di lavoro `Cartel1.xlsx` nella raccolta `Workbooks`. La raccolta `Workbooks` è contenuta nell'oggetto `Application` (ovvero Excel). Scendendo di un altro livello, potete fare riferimento al foglio `Sheet1` di `Cartel1.xlsx` con:

```
Application.Workbooks("Cartel1.xlsx").Worksheets("Sheet1")
```

Scendendo di un altro livello, potete far riferimento a una specifica cella (in questo caso, la cella `A1`) con:

```
Application.Workbooks("Cartel1.xlsx").Worksheets("Sheet1").
     Range("A1")
```

» Se omettete i riferimenti specifici, Excel utilizza gli oggetti attivi. Se `Cartel1. xlsx` è la cartella di lavoro attiva, potete semplificare il riferimento precedente come segue:

```
Worksheets("Sheet1").Range("A1")
```

Se sapete che `Sheet1` è il foglio attivo, potete semplificare ulteriormente il riferimento:

```
Range ("A1")
```

» **Gli oggetti hanno proprietà.** Una proprietà è come un'impostazione di un oggetto. Un oggetto `Range`, per esempio, ha le proprietà `Value` e `Address`. Un oggetto `Chart` ha le proprietà `HasTitle` e `Type`. Potete utilizzare VBA per conoscere e anche per modificare le proprietà di un oggetto.

» **Si fa riferimento a una proprietà di un oggetto combinando il nome dell'oggetto con il nome della proprietà, separati da un punto.** Per esempio, potete fare riferimento alla proprietà `Value` della cella `A1` di `Sheet1` come segue:

```
Worksheets("Sheet1").Range("A1").Value
```

» **Potete assegnare una valore alle variabili.** Una *variabile* è un elemento dotato di nome che memorizza informazioni. Potete usare le variabili nel codice VBA per memorizzare elementi come valori, testi e valori di proprietà. Per assegnare il valore contenuto nella cella `A1` di `Sheet1` alla variabile `Interest`, utilizzate la seguente istruzione VBA:

```
Interest = Worksheets("Sheet1").Range("A1").Value
```

» **Gli oggetti hanno metodi.** Un metodo è un'azione che Excel esegue su un oggetto. Per esempio, uno dei metodi per un oggetto `Range` è `ClearContents`. Questo metodo cancella il contenuto dell'intervallo.

» **Si specifica un metodo concatenando l'oggetto e il metodo, separati da un punto**. Per esempio, la seguente istruzione cancella il contenuto della cella `A1`:

```
Worksheets("Sheet1").Range("A1").ClearContents
```

» **VBA include tutti i costrutti dei moderni linguaggi di programmazione, tra cui variabili, array (che Excel chiama matrici) e cicli.** In altre parole, se siete disposti a dedicare un po' di tempo ad apprendere il linguaggio, potete scrivere codice che fa cose incredibili.

Che lo crediate o no, l'elenco precedente descrive praticamente tutto VBA in poche parole. Ora dovete solo scoprire i dettagli e il significato di quelle parole. Ecco perché questo libro ha bisogno di così tante pagine.

Compatibilità con Excel

RICORDA!

Questo libro è stato scritto per le versioni desktop di Excel 2016 e 2019. Se non disponete di una di queste versioni, correte il rischio di confondervi in alcuni punti.

Se prevedete di distribuire i file Excel/VBA ad altri utenti, è di vitale importanza sapere quali versioni di Excel utilizzano. Le persone che utilizzano vecchie versioni non saranno in grado di sfruttare le funzionalità introdotte nelle versioni successive. Per esempio, se scrivete codice VBA che fa riferimento alla cella XFD1048576 (l'ultima cella di una cartella di lavoro), gli utenti che utilizzano una versione precedente a Excel 2007 otterrano un errore, perché nei fogli di lavoro pre-Excel 2007 vi erano solo 65.536 righe e 255 colonne (l'ultima cella era quindi la IV65536).

Excel 2010 e le versioni successive hanno anche aggiunto alcuni nuovi oggetti, metodi e proprietà. Se li usate nel codice, gli utenti dotati di una versione precedente di Excel otterranno un errore durante l'esecuzione della vostra macro, e non saranno gentili nel dirvelo.

Capitolo 2

Cerchiamo di capirci qualcosa

IN QUESTO CAPITOLO

- **Sviluppare un'utile macro VBA: un esempio pratico, passo dopo passo**
- **Registrare le azioni usando il registratore di macro di Excel**
- **Esaminare e sottoporre a test il codice registrato**
- **Modificare una macro registrata**
- **Trattare i problemi di sicurezza delle macro**

Il modo migliore per entrare nell'acqua fredda è quello di saltarci dentro, senza prolungare l'agonia. Questo capitolo ha più lo scopo di entrare in acqua poco alla volta, senza tuffi di testa.

Al termine di questo capitolo, potreste iniziare a sentirvi a vostro agio con l'argomento della programmazione Excel e sarete felici di aver compiuto questo grande passo. Il capitolo fornisce una dimostrazione dettagliata di che cos'è lo sviluppo di una macro VBA semplice ma utile.

Cominciamo dall'inizio

Prima di definirvi programmatori Excel, dovrete seguire dei riti di iniziazione. Ciò significa che dovrete apportare una piccola modifica, in modo che Excel visualizzi una nuova scheda nella parte superiore dello schermo: Sviluppo. Visualizzare la scheda Sviluppo di Excel è semplice (e poi dovrete farlo una sola volta per tutte). Seguite questi passaggi.

1. **Fate clic destro in qualsiasi parte della barra multifunzione e scegliete dal menu contestuale l'opzione Personalizza barra multifunzione.**
2. **Nella scheda Personalizza barra multifunzione della finestra di dialogo Opzioni di Excel, individuate Sviluppo, nella casella a destra.**
3. **Attivate il segno di spunta accanto a Sviluppo.**
4. **Fate click su OK.**

   Vi ritroverete di nuovo in Excel, con una scheda nuova di zecca: Sviluppo.

FIGURA 2.1 La scheda Sviluppo normalmente è nascosta, ma è facile da trovare.

Che cosa faremo ora

In questo capitolo, creerete la vostra prima macro. Ecco quello che farà.

- Digita il vostro nome in una cella.
- Inserisce la data e l'ora correnti nella cella sottostante.
- Formatta entrambe le celle col grassetto.
- Modifica la dimensione del carattere di entrambe le celle a 16 punti.

Di sicuro non è una macro che possa vincere un premio nella competizione annuale fra programmatori VBA, ma occorre pur iniziare da qualche parte. La macro esegue tutti questi passaggi in un'unica azione. Come vedrete nei paragrafi seguenti, dovrete solo registrare le vostre azioni mentre eseguite questi passaggi. Quindi dovrete sottoporre a test la macro per vedere se funziona. Infine, potrete modificare la macro per aggiungerle alcuni ritocchi. Siete pronti?

Primi passi

Questo paragrafo descrive i passaggi da eseguire *prima* di registrare la macro. In altre parole, sono necessari alcuni preparativi prima di iniziare a divertirsi davvero.

1. **Avviate Excel, se non è già in esecuzione.**
2. **Se necessario, create una nuova cartella di lavoro, vuota.**

   Un modo rapido consiste nel premere Ctrl + N.
3. **Fate clic sulla scheda Sviluppo e individuate il pulsante Usa riferimenti relativi nel gruppo Codice.**

   Se il colore del pulsante è già diverso da quello degli altri pulsanti, siete nella modalità corretta. Se il pulsante Usa riferimenti relativi è dello stesso colore degli altri, dovrete fare clic su di esso per abilitare questa opzione.

Esploreremo l'uso del pulsante Usa riferimenti relativi nel Capitolo 6. Per ora, assicuratevi solo che l'opzione sia attivata. Quando è acceso, il pulsante Usa riferimenti relativi è di un colore diverso.

Registrazione della macro

Ecco la parte pratica. Seguite attentamente queste istruzioni.

1. **Selezionate una cella.**

   Una cella qualsiasi andrà bene.
2. **Selezionate Sviluppo ➪ Codice ➪ Registra macro o fate clic sul pulsante di registrazione macro sulla barra di stato.**

   Verrà visualizzata la finestra di dialogo Registra macro, rappresentata nella Figura 2.2.
3. **Inserite un nome per la macro.**

   Excel suggerisce un nome predefinito (qualcosa come *Macro1*), ma è meglio usare un nome più descrittivo. *NameAndTime* (senza spazi) è un buon nome per questa macro.
4. **Fate clic sulla casella Tasto di scelta rapida e immettete, come tasto di scelta rapida, Maiusc + N (con la "n" maiuscola).**

   La specifica di un tasto di scelta rapida è facoltativa. Se ne specificate uno, potrete eseguire la macro premendo la combinazione di tasti scelta, in questo caso, Ctrl + Maiusc + N. Notate che se assegnate un tasto di scelta rapida già in uso (per esempio Ctrl + C), perderete la normale funzionalità di tale combinazione di tasti; al suo posto Excel richiamerà la vostra macro.
5. **Assicuratevi che l'impostazione Memorizza macro in sia su Questa cartella di lavoro.**
6. **Potete anche scrivere del testo nella casella Descrizione.**

   Questo passaggio è facoltativo. Ad alcune persone piace descrivere quello che fa (o si *suppone* che faccia) la macro.
7. **Fate click su OK.**

   La finestra di dialogo Registra macro si chiuderà e si attiverà il registratore di macro di Excel. Da questo momento in poi, Excel controllerà tutto ciò che fate e lo convertirà in codice VBA.
8. **Digitate il vostro nome nella cella attiva.**
9. **Spostate il cursore nella cella sottostante e inserite questa formula:**

   ```
   = ADESSO()
   ```

   La formula visualizza la data e l'ora correnti.
10. **Selezionate la cella della formula e premete Ctrl + C per copiarla negli Appunti.**
11. **Scegliete Home ➪ Appunti ➪ Incolla ➪ Valori (V).**

    Questo comando converte la formula nel suo valore.
12. **Con la cella della data selezionata, premete Maiusc + freccia su per selezionare quella cella e quella sovrastante (quella che contiene il vostro nome).**
13. **Usate i controlli presenti nel gruppo Home ➪ Carattere per modificare la formattazione in grassetto e dimensione del carattere pari a 16 punti.**
14. **Selezionate Sviluppo ➪ Codice ➪ Interrompi registrazione.**

Il registratore di macro verrà fermato.

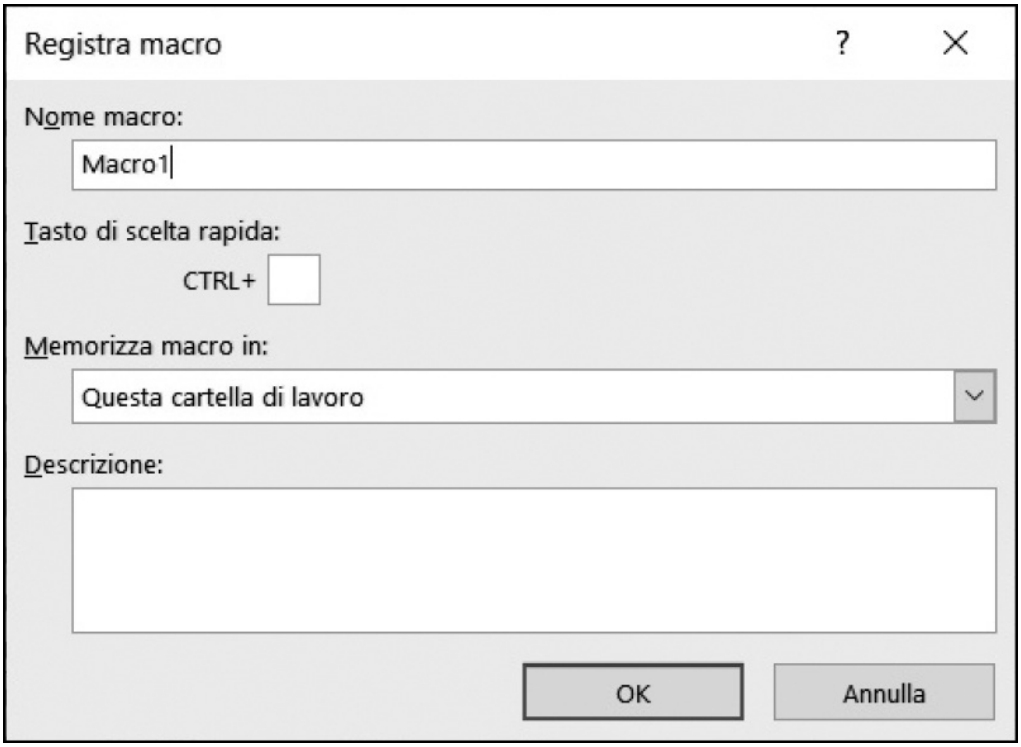

FIGURA 2.2 Quando state per registrare una macro, appare la finestra di dialogo Registra macro.

Congratulazioni! Avete appena creato la vostra prima macro VBA. Forse è il caso di telefonare a qualcuno per comunicare la buona notizia.

Test della macro

Ora potete provare a usare questa macro per vedere se funziona correttamente. Per sottoporre a test la macro, spostatevi in una cella vuota e premete Ctrl + Maiusc + N (o comunque la combinazione di tasti che le avete assegnato).

Excel eseguirà la macro in un lampo. Al termine troverete visualizzati a caratteri cubitali e in grassetto il vostro nome e la data e l'ora correnti.

CONSIGLIO

Un altro modo per eseguire la macro è selezionare Sviluppo ➪ Codice ➪ Macro (o premere Alt + F8) per visualizzare la finestra di dialogo Macro. Selezionate la macro dall'elenco (in questo caso, *NameAndTime*) e fate clic su Esegui. Assicuratevi di selezionare la cella che ospiterà il vostro nome, prima di eseguire la macro.

Esaminiamo la macro

OK, avete registrato una macro e l'avete anche provata. Se siete tipi curiosi, probabilmente vi starete chiedendo che aspetto abbia questa macro. E potreste perfino chiedervi dove sia stata memorizzata.

Vi ricordate quando avete iniziato a registrare la macro? Avete indicato a Excel di memorizzare la macro in Questa cartella di lavoro. La macro viene archiviata all'interno della cartella di lavoro, ma per vederne l'aspetto è necessario attivare Visual Basic Editor (VBE, per gli amici).

Seguite questi passaggi per vedere la macro.

1. **Scegliete Sviluppo ➪ Codice ➪ Visual Basic (o premete Alt + F11).**

   Viene visualizzata la finestra del programma Visual Basic Editor, rappresentata nella Figura 2.3. Questa finestra è altamente personalizzabile, quindi la vostra potrebbe avere un aspetto un po' diverso. La finestra di VBE contiene diverse altre finestre e probabilmente incute un certo timore. Ma non agitatevi: vi ci abituerete.

2. **Nella finestra di VBE, individuate il riquadro Progetto.**

Il riquadro Progetto (chiamato anche Gestione progetti) contiene un elenco di tutte le cartelle di lavoro e di tutti i componenti aggiuntivi attualmente aperti. Ogni progetto è organizzato come un albero e può essere espanso (per mostrare più informazioni) o ridotto (per mostrarne meno).

CONSIGLIO

VBE utilizza diverse finestre, ognuna delle quali può essere aperta o chiusa. Se una finestra non è immediatamente visibile in VBE, potete scegliere un'opzione dal menu Visualizza per visualizzarla. Per esempio, se il riquadro Progetto non è visibile, potete scegliere Visualizza ➪ Gestione progetti (o premere Ctrl + R) per visualizzarlo. Potete visualizzare qualsiasi altra finestra di VBE in modo analogo. I componenti del VBE sono trattati nel Capitolo 3.

3. **Selezionate il progetto che corrisponde alla cartella di lavoro nella quale avete registrato la macro.**

   Se non avete ancora salvato la cartella di lavoro, il progetto sarà probabilmente chiamato VBAProject (Cartel1).

4. **Fate clic sul segno più (+) a sinistra della cartella Moduli.**

   L'albero si espanderà per mostrare il contenuto del Modulo1, che è l'unico modulo nel progetto.

5. **Fate doppio clic su Modulo1.**

   Il codice VBA di quel modulo verrà visualizzato in una finestra di codice (vedere la Figura 2.3). Lo schermo potrebbe non apparire esattamente come nella Figura 2.3. Il codice registrato dipende dalle azioni specifiche che avete eseguito durante la registrazione della macro.

FIGURA 2.3 VBE visualizza il codice VBA nel Modulo1 di Cartel1.

A questo punto, il contenuto della macro probabilmente vi sembrerà greco antico. Non preoccupatevi. Nei vari capitoli che incontrerete lungo il cammino tutto vi diventerà più chiaro, come la vista dal monte Olimpo.

La macro *NameAndTime* è composta da diverse istruzioni. Excel le esegue una alla volta, dall'alto verso il basso. Un'istruzione preceduta da un apostrofo (') è un commento. I commenti sono inclusi solo per vostra informazione e vengono ignorati da Excel. In altre parole, Excel non li considera proprio, i commenti.

MA QUESTO IO NON L'HO REGISTRATO!

Il registratore di macro registra proprio come un registratore di suoni. Quando riproducete una registrazione e ascoltate la vostra voce, invariabilmente vi viene da dire "Ma questa non sembra la mia voce". E quando guardate la macro che avete registrato, potreste vedere alcune azioni che non pensavate di aver registrato.

Nel registrare l'esempio *NameAndTime*, avete modificato solo le dimensioni del carattere, tuttavia il codice registrato mostra tutti i tipi di istruzioni che modificano i caratteri: `Strikethrough` (barrato), `Supercript` (apice), `Shadow` (ombra) e così via. Non temete; capita sempre. Excel registra molto codice apparentemente inutile. Nei prossimi

capitoli scopriremo come eliminare le opzioni aggiuntive da una macro registrata.

Modificare la macro

Come potete immaginare, VBE non vi permette solo di visualizzare la vostra macro, ma anche di modificarla. Anche se probabilmente al momento non avete alcuna idea di cosa potreste fare, queste modifiche sono facili da eseguire.

- Modificate il nome che viene inserito nella cella attiva. Se avete un cane, potete usare il suo nome.
- Modificate il tipo o le dimensioni del tipo di carattere.
- Cercate di trovare la posizione appropriata per questa nuova istruzione, che mette le celle in corsivo:

```
Selection.Font.Italic = True
```

CONSIGLIO

Lavorare su un frammento di codice VBA è un po' come lavorare su un documento testuale (tranne per il fatto che il testo non va a capo e non potete formattarlo). A pensarci bene, è più simile al Blocco note di Windows. Potete premere Invio per iniziare una nuova riga e i tasti di editing funzionano come previsto.

Dopo aver apportato le modifiche, tornate a Excel e provate la macro modificata, per vedere come funziona. Così come potete premere Alt + F11 in Excel per visualizzare il VBE, potete premere Alt + F11 nel VBE per tornare a Excel.

Salvare le cartelle di lavoro contenenti macro

Se in una cartella di lavoro memorizzate una o più macro, il file deve essere salvato come tipo di file speciale, con abilitazione delle macro. In altre parole, il file deve essere salvato con estensione XLSM anziché XLSX.

Per esempio, quando salvate la cartella di lavoro che contiene la macro *NameAndTime*, nella finestra di dialogo Salva con nome troverete per impostazione predefinita l'impostazione XLSX (un formato che *non* può contenere macro). Se non modificate il formato del file in XLSM, Excel visualizzerà l'avviso mostrato nella Figura 2.4. Dovrete fare clic su No, e poi selezionare Cartella di lavoro con attivazione macro di Excel (*.xlsm) dall'elenco a discesa del tipo di file.

FIGURA 2.4 Se la cartella di lavoro contiene macro e tentate di salvarla in un formato senza macro, Excel vi avviserà.

La sicurezza delle macro

La sicurezza delle macro è una funzionalità chiave di Excel. Questo perché VBA è un linguaggio potente, talmente potente che potete creare una macro che può danneggiare seriamente il computer. Una macro può eliminare file, inviare informazioni ad altri computer e perfino danneggiare Windows al punto da impedire l'avvio del sistema.

Le funzionalità di protezione introdotte fin da Excel 2007 sono state concepite proprio per prevenire questi tipi di problemi.

La Figura 2.5 mostra la sezione Impostazioni delle macro della finestra di dialogo Centro protezione. Per visualizzare questa finestra di dialogo, selezionare Sviluppo ➪ Codice ➪ Sicurezza macro.

FIGURA 2.5 La sezione Impostazioni delle macro della finestra di dialogo Centro protezione.

Per impostazione predefinita, Excel utilizza l'opzione Disabilita tutte le macro con notifica. Con questa impostazione attiva, se aprite una cartella di lavoro che contiene delle macro (e il file non è "firmato" digitalmente o salvato in una posizione attendibile), Excel visualizza un avviso simile a quello rappresentato nella Figura 2.6. Se siete sicuri che la cartella di lavoro provenga da una fonte attendibile, fate clic su Abilita macro e le macro saranno abilitate.

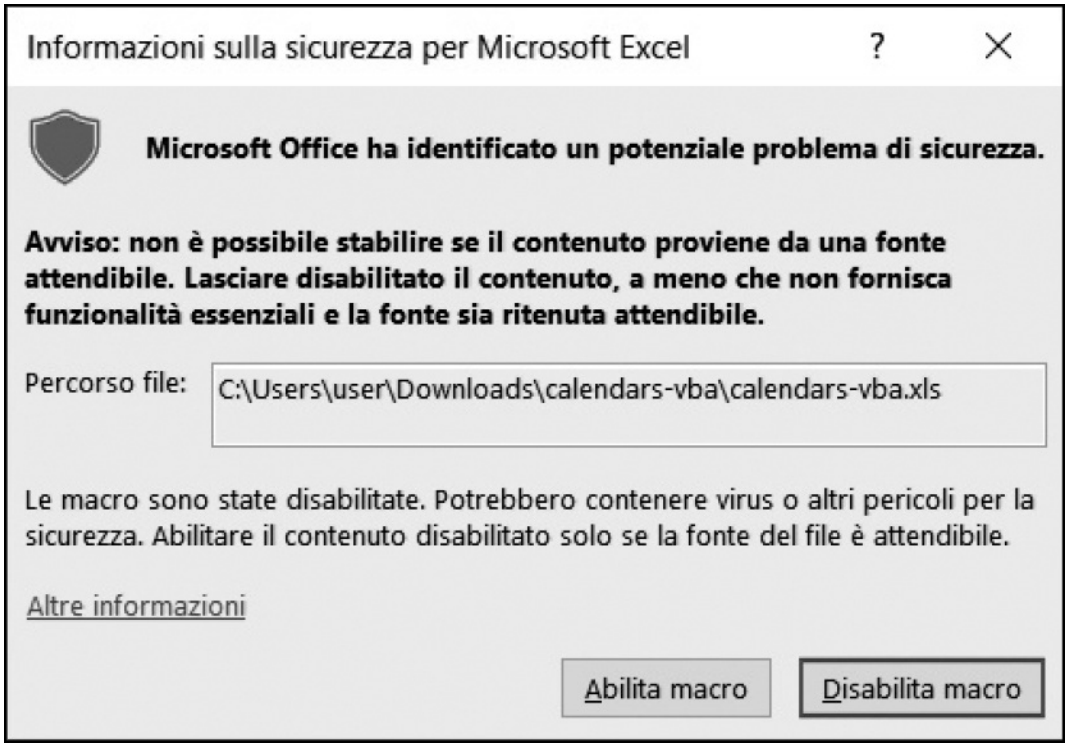

FIGURA 2.6 Excel avverte che il file da aprire contiene macro.

RICORDA!

La finestra rappresentata nella Figura 2.6 viene visualizzata solo se VBE è aperto. Altrimenti, Excel visualizza un avviso di sicurezza sopra la barra della formula, mostrato nella Figura 2.7. Se siete certi che la cartella di lavoro sia sicura, fate clic sul pulsante Abilita contenuto per abilitare le macro. Per utilizzare la cartella di lavoro senza macro, fare clic sulla X a destra, per eliminare l'avviso.

FIGURA 2.7 Excel vi avverte che la cartella di lavoro aperta contiene macro. Questo è l'avviso visualizzato quando VBE non è aperto.

Excel ricorda se avete indicato che una cartella di lavoro è sicura. Quindi la prossima volta che la aprirete non visualizzerà più questo avviso di sicurezza.

Forse il modo migliore per gestire la sicurezza delle macro è indicare, come *percorsi attendibili*, una o più cartelle. Tutte le cartelle di lavoro in una posizione attendibile vengono aperte senza alcun avviso. Le cartelle attendibili possono essere indicate nella sezione Posizioni attendibili della finestra di dialogo Centro protezione.

Se desiderate scoprire quali sono le altre impostazioni di protezione delle macro, premete F1 mentre è visualizzata la sezione Impostazioni delle macro della finestra di dialogo Centro protezione. Verrà visualizzata una schermata della Guida che descrive le impostazioni di sicurezza.

Qualcosa di più sulla macro NameAndTime

Al termine di questo libro, sarete assolutamente in grado di comprendere come funziona la macro *NameAndTime* e sarete anche in grado di sviluppare macro ben più sofisticate. Per ora, questo capitolo termina con alcuni spunti aggiuntivi sulla macro.

- Affinché questa macro funzioni, è necessario che la cartella di lavoro sia aperta. Chiudendo la cartella di lavoro, la macro non sarà più disponibile, quindi non potrà più funzionare (e la sua combinazione di tasti non avrà più alcun effetto).
- Finché la cartella di lavoro contenente la macro è aperta, potrete eseguire la macro anche in un'altra cartella di lavoro. In altre parole, non è affatto necessario che la cartella di lavoro contenente la macro sia quella attiva.
- La macro non è codice davvero "intelligente". Sovrascriverà il testo esistente senza alcun preavviso e i suoi effetti non possono essere annullati.
- Prima di iniziare a registrare la macro, le avete assegnato un Tasto di scelta rapida. Questo è solo uno dei tanti modi per eseguire la macro (ne scoprirete altri nel Capitolo 5).
- Potete creare questa macro anche manualmente, senza registrarla. Ma per farlo dovete avere una buona conoscenza del linguaggio VBA (siate pazienti, ci arriverete presto).
- Potete memorizzare questa macro nella cartella di lavoro Macro personale. In tal caso, la macro sarà automaticamente disponibile ogni volta che avvierete Excel (per ulteriori dettagli sulla cartella di lavoro Macro personale consultate il Capitolo 6).
- Potete anche convertire la cartella di lavoro in un file di componenti aggiuntivi (ulteriori informazioni nel Capitolo 21).

Congratulazioni! Siete ormai entrati nel club dei programmatori Excel (e… no, non c'è nessuna stretta di mano segreta o anello decodificatore).

2

Il funzionamento di VBA in Excel

IN QUESTA PARTE...

- Accedere alle parti più importanti di Visual Basic Editor.
- I moduli di codice VBA (dove memorizzate il codice VBA).
- Panoramica sul modello a oggetti di Excel.
- Due concetti chiave: le proprietà e i metodi degli oggetti.
- Differenze tra routine Sub e routine Function.
- Corso accelerato sull'utilizzo del registratore di macro di Excel.

Capitolo 3

Usare Visual Basic Editor

IN QUESTO CAPITOLO

- **L'editor di Visual Basic**
- **Le varie parti di Visual Basic Editor**
- **Il contenuto di un modulo VBA**
- **Tre modi per inserire codice VBA in un modulo**
- **Personalizzare l'ambiente VBA**

Come utenti Excel più esperti della media, probabilmente già conoscete una buona quantità di cose su cartelle di lavoro, formule, grafici e altri extra di Excel. Ora è il momento di espandere i vostri orizzonti ed esplorare un suo aspetto completamente nuovo: Visual Basic Editor. In questo capitolo, scoprirete come lavorare con Visual Basic Editor, e proverete a scrivere dei programmi VBA.

Che cos'è Visual Basic Editor?

L'editor di Visual Basic (spesso abbreviato con VBE) è un'applicazione a sé stante dedicata alla scrittura e all'editing di macro VBA. A partire da Excel 2013, ogni cartella di lavoro viene visualizzata in una finestra separata. Tuttavia, esiste una sola finestra di VBE, che funziona con tutte le finestre aperte di Excel.

RICORDA!

Non potete eseguire VBE separatamente; per poter richiamare VBE, Excel deve necessariamente essere in esecuzione.

Attivare VBE

Il modo più rapido per attivare VBE è premere Alt + F11 da Excel. Per tornare a Excel, basta premere nuovamente Alt + F11, oppure potete semplicemente fare clic sul pulsante Chiudi sulla barra del titolo di VBE. Quando si chiude la finestra di VBE, Excel viene avviato.

Potete avviare VBE anche selezionando il comando Sviluppo selezionando il comando Sviluppo ➪ Codice ➪ Visual Basic. Se nella parte superiore della finestra di Excel non trovate la scheda Sviluppo, consultate il Capitolo 2, dove spiego come visualizzarla.

I vari componenti di VBE

La Figura 3.1 mostra il programma VBE, con evidenziati alcuni dei suoi principali componenti.

FIGURA 3.1 VBE è ampiamente personalizzabile.

La vostra finestra di VBE potrebbe non somigliare esattamente a quella rappresentata nella Figura 3.1. L'editor VBE contiene diverse finestre ed è altamente personalizzabile. Potete nascondere, ridisporre, aggangiare le sue finestre come volete.

Barra dei menu

La barra dei menu di VBE funziona esattamente come ogni altra barra dei menu. Contiene i comandi necessari per eseguire operazioni con i vari componenti di VBE. A molti dei comandi del menu sono associati dei tasti di scelta rapida.

CONSIGLIO

VBE offre anche dei menu contestuali. Con un clic destro su un qualsiasi oggetto in VBE otterrete il menu contestuale, contenente i comandi più comuni.

Barra degli strumenti

La barra degli strumenti Standard, posta per impostazione predefinita appena sotto la barra dei menu (vedi Figura 3.1), è una delle quattro barre degli strumenti disponibili in VBE. Potete personalizzare le barre degli strumenti, spostarle, visualizzarne altre e così via. Se desiderate, per esplorarle selezionate Visualizza ➪ Barre degli strumenti. La maggior parte delle persone lascia la barra degli strumenti così com'è.

Riquadro Progetto

Il riquadro Progetto mostra uno schema ad albero che elenca ogni cartella di lavoro attualmente aperta in Excel (inclusi i componenti aggiuntivi e le cartelle di lavoro nascoste). Fate doppio clic sugli elementi per espanderli o contrarli. Per ulteriori dettagli consultate il prossimo paragrafo "Il riquadro Progetto".

Se il riquadro Progetto non è visibile, premete Ctrl + R o scegliete Visualizza ➪ Gestione progetti. Per nascondere il riquadro Progetto, fate clic sul pulsante Chiudi sulla barra del titolo oppure fate clic destro in qualsiasi punto del riquadro Progetto e scegliete dal menu contestuale l'opzione Nascondi.

Finestra del codice

La finestra del codice è il luogo in cui inserite il codice VBA. Ogni oggetto, all'interno di un progetto, ha una sua finestra di codice. Per visualizzare la finestra Codice, fate doppio clic sull'oggetto nel riquadro Progetto. Per esempio, per visualizzare la finestra Codice per l'oggetto Foglio1 in Cartel1, fate doppio clic su Foglio1 sotto VBAProject (Cartel1). Se non avete ancora aggiunto codice VBA, la finestra Codice sarà vuota.

Per ulteriori informazioni sulle finestre di codice, consultate il paragrafo "Usare la finestra Codice" più avanti in

questo capitolo.

Finestra Immediata

La finestra Immediata potrebbe essere visibile o meno. Se non è visibile, premete Ctrl + G o scegliete Visualizza ➪ Finestra Immediata. Per chiudere la finestra Immediata, fate clic sul pulsante Chiudi sulla barra del titolo (o fate clic con il pulsante destro in qualsiasi punto della finestra Immediata e scegliete dal menu contestuale l'opzione Nascondi).

CHE COSA C'È DI NUOVO NELL'EDITOR DI VISUAL BASIC?

Il programma Excel 2007 ha introdotto un'interfaccia utente nuova di zecca. Menu e barre degli strumenti sono stati sostituiti con una nuova interfaccia utente a barra multifunzione. Ma il VBE non ha mai ricevuto questo tipo di *restyling* e ha mantenuto i menu tradizionali, con allegata barra degli strumenti.

Il linguaggio di programmazione VBA è stato aggiornato per accogliere le nuove specifiche di Excel, ma non è cambiato nient'altro.

Una cosa è cambiata: il sistema della Guida. In passato, le informazioni della Guida erano memorizzate sul computer e avevate solo come opzione l'accesso alla Guida via Internet. A partire da Excel 2013, tutte le informazioni della Guida sono disponibili online e vengono visualizzate nel browser web. In altre parole, per accedere alla Guida è necessario essere connessi a Internet. Tuttavia, potete anche scaricare una vostra copia locale della Guida dal sito di Microsoft. Fate una ricerca sul Web dei termini "scaricare la documentazione di Excel VBA".

La finestra Immediata è molto utile per l'esecuzione diretta delle istruzioni VBA e per il debug del codice. Ma se state solo iniziando a usare VBA, questa finestra non vi sarà poi così utile, quindi sentitevi liberi di nasconderla per liberare spazio sullo schermo per altre cose.

Il Capitolo 13 illustra in dettaglio l'uso della finestra Immediata. La troverete molto utile, più avanti!

Il riquadro Progetto

In VBE, ogni cartella di lavoro di Excel e ogni componente aggiuntivo aperto è un progetto. Potete pensare a un progetto come a una collezione di oggetti disposti secondo una struttura gerarchica. Potete espandere un progetto facendo clic sul segno più (+) a sinistra del nome nel riquadro Progetto. Potete comprimere un progetto facendo clic sul segno meno (-) a sinistra del nome. Oppure potete fare doppio clic sugli elementi per espanderli.

RICORDA!

Se un progetto è protetto da password, facendo doppio clic sul suo nome vi verrà richiesta la password. Se non la conoscete, non potrete espandere il progetto, il che significa che non potrete visualizzare o modificare nessuna parte del progetto stesso.

La Figura 3.2 mostra un riquadro Progetto con l'elenco di quattro progetti: un componente aggiuntivo denominato Pup7.xlam, una cartella di lavoro non salvata denominata Cartel1, una cartella di lavoro denominata Investimenti.xlsm e la cartella di lavoro Cartella macro personale (che viene sempre denominata PERSONAL.XLSB). Dei quattro, solo il progetto Investimenti.xlsm è espanso per mostrare tutti i suoi oggetti.

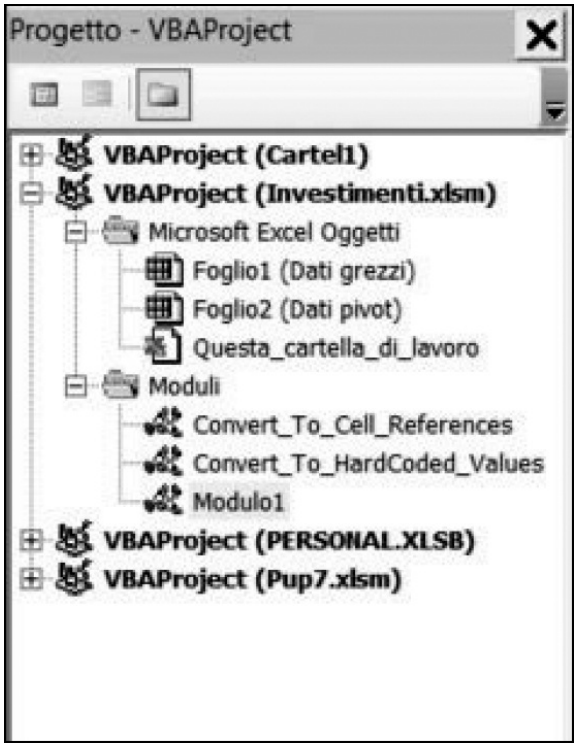

FIGURA 3.2 Questa finestra elenca quattro progetti. Uno di questi è espanso, per mostrare i suoi oggetti.

Ogni progetto si espande per mostrare almeno un nodo chiamato Microsoft Excel Oggetti. Questo nodo si espande per mostrare un elemento per ogni foglio nella cartella di lavoro (ogni foglio è considerato un oggetto) e un altro oggetto chiamato Questa_cartella_di_lavoro (che rappresenta l'oggetto Workbook). Se il progetto ha moduli VBA, l'elenco dei progetti mostra anche un nodo Moduli. E come scoprirete nella Parte 4, un progetto può contenere anche un nodo chiamato Form, che contiene oggetti UserForm (sono finestre di dialogo personalizzate).

In questa fase il concetto di *oggetto* potrebbe esservi poco chiaro. Ma non temete. Le cose diventeranno molto più chiare nei prossimi capitoli. Non preoccupatevi se non capite proprio tutto, in questa fase.

Aggiungere un nuovo modulo VBA

Ecco come potete aggiungere a un progetto un nuovo modulo VBA.

1. **In VBE, selezionate il nome del progetto nel riquadro Progetto.**
2. **Scegliete Inserisci ➪ Modulo.**

Oppure

1. **Fate clic destro sul nome del progetto.**
2. **Scegliete Inserisci ➪ Modulo dal menu contestuale.**

CONSIGLIO

Quando registrate una macro, Excel crea automaticamente un modulo VBA che conterrà il codice registrato. Quale cartella di lavoro contiene il modulo per la macro registrata dipende da dove avete scelto di memorizzare la macro registrata, appena prima di iniziare la registrazione.

Rimuovere un modulo VBA

A volte, è anche necessario rimuovere un modulo VBA da un progetto. Per esempio, il modulo potrebbe contenere codice non più necessario oppure è vuoto, perché avete inserito un modulo ma poi avete cambiato idea. Ecco come rimuovere un modulo VBA da un progetto.

1. **Selezionate il nome del modulo nel riquadro Progetto.**
2. **Selezionate il comando File ➪ Rimuovi *xxx*, dove *xxx* è il nome del modulo.**

Oppure

1. **Fate clic destro sul nome del modulo.**

2. **Scegliete Rimuovi *xxx* dal menu contestuale.**

   Excel, cerca sempre di impedirvi di fare qualcosa di cui potreste pentirvi, e così vi chiederà se desiderate esportare il codice contenuto nel modulo, prima di eliminarlo. Quasi sempre non lo si fa (se volete esportare il modulo, consultate il prossimo paragrafo).

Potete rimuovere i moduli VBA, ma non potete rimuovere gli altri moduli di codice, quelli per gli oggetti Foglio o Questa_cartella_di_lavoro.

Esportare e importare oggetti

In un progetto VBA, ogni oggetto può essere salvato in un file a sé. Il salvataggio di un singolo oggetto in un progetto è chiamato *esportazione*. Analogamente potete anche *importare nuovi* oggetti in un progetto. L'esportazione e importazione di oggetti può essere utile se desiderate utilizzare un oggetto (come un modulo VBA o uno UserForm) in un altro progetto. O magari dovete inviare a un collega una copia di un modulo VBA, e questo collega potrà importarlo nel suo progetto.

Per esportare un oggetto procedete come segue.

1. **Selezionate un oggetto nel riquadro Progetto.**
2. **Selezionate il comando File ➪ Esporta file o premete Ctrl + E.**

   Si aprirà una finestra di dialogo che richiede un nome di file. Notate che l'oggetto rimane nel progetto; ne viene solo esportata una copia. Excel aggiungerà al file l'estensione corretta, che dipende dal tipo di oggetto che state esportando. In ogni caso, il risultato è un file di testo. Se siete curiosi di esplorarne il contenuto, potete aprirlo con un qualsiasi editor di testi e dargli un'occhiata.

Per importare un file in un progetto procedete come segue.

1. **Selezionate il nome del progetto nel riquadro Progetto.**
2. **Selezionate il comando File ➪ Importa file o premete Ctrl + M.**

   Si aprirà una finestra di dialogo che richiede un file.
3. **Individuate il file e fate clic su Apri.**

RICORDA!

Dovreste importare file di oggetti VBA solo se ne conoscete la provenienza. Altrimenti, potreste introdurre nel sistema macro che eseguono azioni pericolose per il sistema.

Usare la finestra Codice

RICORDA!

Mano a mano che diverrete esperti di VBA, trascorrerete anche parecchio tempo a lavorare nelle finestre di codice. Le macro che registrate vengono memorizzate in un modulo, ma potete anche digitare il codice VBA direttamente in un modulo VBA.

Ridurre a icona e massimizzare le finestre

Se avete aperto più progetti, il VBE potrebbe aver aperto, contemporaneamente, più finestre di codice. La Figura 3.3 mostra un esempio.

FIGURA 3.3 Non è mai bello vedere un eccesso di finestre di codice.

Le finestre di codice sono simili alle finestre delle cartelle di lavoro in Excel. Potete ridurle a icona, ingrandirle, ridimensionarle, nasconderle, riordinarle e così via. Molti preferiscono ingrandire al massimo la finestra di codice sulla quale stanno lavorando. In questo modo possono visualizzare più codice, senza distrarsi.

Per ingrandire una finestra di codice, fate clic sul pulsante Ingrandisci nella sua barra del titolo (accanto alla X), oppure fate direttamente doppio clic sulla barra del titolo. Per ripristinare una finestra di codice alle sue dimensioni originali, fate clic sul pulsante Ripristina finestra. Quando una finestra è ingrandita, la sua barra del titolo non è visibile, quindi troverete il pulsante Ripristina finestra sotto la barra del titolo di VBE.

A volte, potreste voler visualizzare insieme due o più finestre di codice. Per esempio, potreste voler confrontare il codice di due moduli o copiare il codice da un modulo a un altro. Potete disporre le finestre manualmente o selezionare il comando Finestra ➪ Affianca orizzontalmente o Finestra ➪ Affianca vericicalmente, per disporle automaticamente.

Potete passare rapidamente da una finestra di codice all'altra premendo Ctrl + F6. Ripetendo questa combinazione di tasti, continuerete a passare da una finestra all'altra, tra tutte le finestre di codice attualmente aperte. Premendo Ctrl + Maiusc + F6 passerete da una finestra all'altra in ordine inverso.

CONSIGLIO

Riducendo a icona una finestra Codice la togliete di mezzo. Potete anche fare clic sul pulsante Chiudi della finestra (che visualizza X) sulla barra del titolo di una finestra di codice, per chiuderla (la chiusura di una finestra la nasconde e basta, non perderete nulla). Per riaprirla, fate doppio clic sull'oggetto appropriato nel riquadro Progetto. Lavorare con le finestre di codice, comunque, sembra più difficile di quanto non sia in realtà.

Creare un modulo

In generale, un modulo VBA può contenere tre tipi di codice.

- **Dichiarazioni:** una o più istruzioni informative che fornite a VBA. Per esempio, potete dichiarare il tipo di dati delle variabili che prevedete di utilizzare o potete impostare delle opzioni valide per l'intero modulo. Le dichiarazioni sono fondamentalmente istruzioni gestionali. Non vengono effettivamente eseguite.
- **Routine Sub:** un insieme di istruzioni che, se eseguite, svolgono alcune azioni.
- **Routine Function:** un insieme di istruzioni che restituiscono un valore (è un po' il concetto su cui si basano le funzioni del foglio di lavoro, per esempio SOMMA).

Un singolo modulo VBA può contenere un numero arbitrario di routine Sub, routine Function e dichiarazioni. Esiste un solo limite: circa 64.000 caratteri per modulo. È davvero improbabile che vi avvicinerete a tale limite. Ma nel caso in cui questo avvenga, la soluzione è semplice: basta creare un nuovo modulo. Il modo in cui organizzate un modulo VBA dipende completamente da voi. Alcuni preferiscono mantenere tutto il codice VBA di un'applicazione in un singolo modulo VBA; ad altri piace suddividere il codice in più moduli. È una scelta personale, proprio come l'organizzazione dell'arredamento di casa.

Inserire codice VBA in un modulo

Un modulo VBA vuoto è come il cibo finto nelle vetrine di alcuni ristoranti cinesi: sembra commestibile, ma in realtà non lo è. Prima di poter fare qualcosa di significativo, occorre inserire del codice VBA nel modulo. Potete farlo in tre modi.

- Inserire il codice direttamente.
- Utilizzare il registratore di macro di Excel per registrare le azioni e convertirle in codice VBA (vedere il Capitolo 6).
- Copiare il codice da un modulo e incollarlo in un altro.

PER I PIÙ CURIOSI

BREAK TERMINOLOGICO

In questo libro, noterete che vengono usati i termini *procedura Sub*, *routine*, *programma*, *procedura* e *macro*. Questa abbondanza di termini può confondere. I programmatori di solito usano la parola *procedura* per descrivere un'attività automatizzata. Tecnicamente, una procedura può essere di tipo Sub o Function, entrambe a volte chiamate *routine* e perfino *programmi*. Tutti questi termini sono usati un po' come sinonimi. Come vedrete nei prossimi capitoli, tuttavia, vi è un'importante differenza tra le routine Sub e Function. Per ora, non preoccupatevi troppo della terminologia. L'importante è impradronirsi dei concetti.

Inserire direttamente il codice

A volte, la via migliore è quella più diretta. Inserire il codice direttamente vuol dire... scrivere direttamente il codice. In altre parole, dovrete digitare il codice utilizzando la tastiera. L'inserimento e l'editing del testo in un modulo VBA funziona esattamente come vi immaginate. Potete selezionare, copiare, tagliare, incollare ed eseguire tutte le altre classiche operazioni sul testo.

Potete usare il tasto Tab per indentare alcune righe e rendere il codice più leggibile. L'indentazione non è obbligatoria, ma è una buona abitudine. Studiando il codice di questo libro, capirete perché è utile indentare le righe di codice.

RICORDA!

Una singola riga di codice VBA può essere arbitrariamente lunga. Tuttavia, potete utilizzare i caratteri di continuazione della riga per suddividere le righe di codice più lunghe. Per far continuare una singola riga di codice (ovvero una singola istruzione) da una riga di testo a quella successiva, basta usare uno spazio seguito da un carattere di sottolineatura (_). A questo punto potete proseguire l'istruzione sulla riga successiva. Non dimenticatevi lo spazio. Un carattere di sottolineatura che non sia preceduto da uno spazio non basterà.

Ecco un esempio di una singola istruzione sudivisa su tre righe di testo:

```
Selection.Sort Key1:=Range("A1"), _
     Order1:=xlAscending, Header:=xlGuess, _
     Orientation:=xlTopToBottom
```

Questa istruzione funzionerebbe esattamente nello stesso modo se fosse stata inserita su una singola riga (senza i caratteri di continuazione di riga). Notate che la seconda e la terza riga di questa istruzione sono indentate. Il rientro è facoltativo, ma aiuta a chiarire il fatto che queste righe non sono istruzioni distinte.

CONSIGLIO

Gli ingegneri che hanno progettato VBE hanno previsto che avremmo commesso errori. Pertanto, il VBE offre più livelli di annullamento e ripetizione. Se per sbaglio avete cancellato un'istruzione, fate clic sul pulsante Annulla sulla barra degli strumenti (o premete Ctrl + Z) fino a quando l'istruzione non tornerà al suo posto. Dopo aver

annullato, potete fare clic sul pulsante Ripristina per riapplicare le modifiche annullate. Vi sentite pronti per inserire del vero codice VBA? Provate i seguenti passaggi.

1. **Create una nuova cartella di lavoro in Excel.**
2. **Premete Alt + F11 per attivare il VBE.**
3. **Fate clic sul nome della nuova cartella di lavoro nel riquadro Progetto.**
4. **Scegliete Inserisci ➪ Modulo per inserire nel progetto un nuovo modulo VBA.**
5. **Digitate nel modulo il seguente codice:**

```
Sub GuessName()
      Msg = "Il tuo nome è " & Application.UserName & "?"
      Ans = MsgBox(Msg, vbYesNo)
      If Ans = vbNo Then MsgBox "Chiedo perdono."
      If Ans = vbYes Then MsgBox "Sono un fenomeno!"
End Sub
```

6. **Posizionate il cursore in un punto qualsiasi all'interno del testo che avete digitato e premete F5 per eseguire la routine.**

   F5 è una scorciatoia per Esegui ➪ Esegui Sub/UserForm. Se avete inserito il codice correttamente, Excel eseguirà la routine e potrete rispondere alla semplice finestra di dialogo rappresentata nella Figura 3.4. Il testo presentato nella finestra di dialogo sarà ovviamente diverso da quello mostrato nella figura.

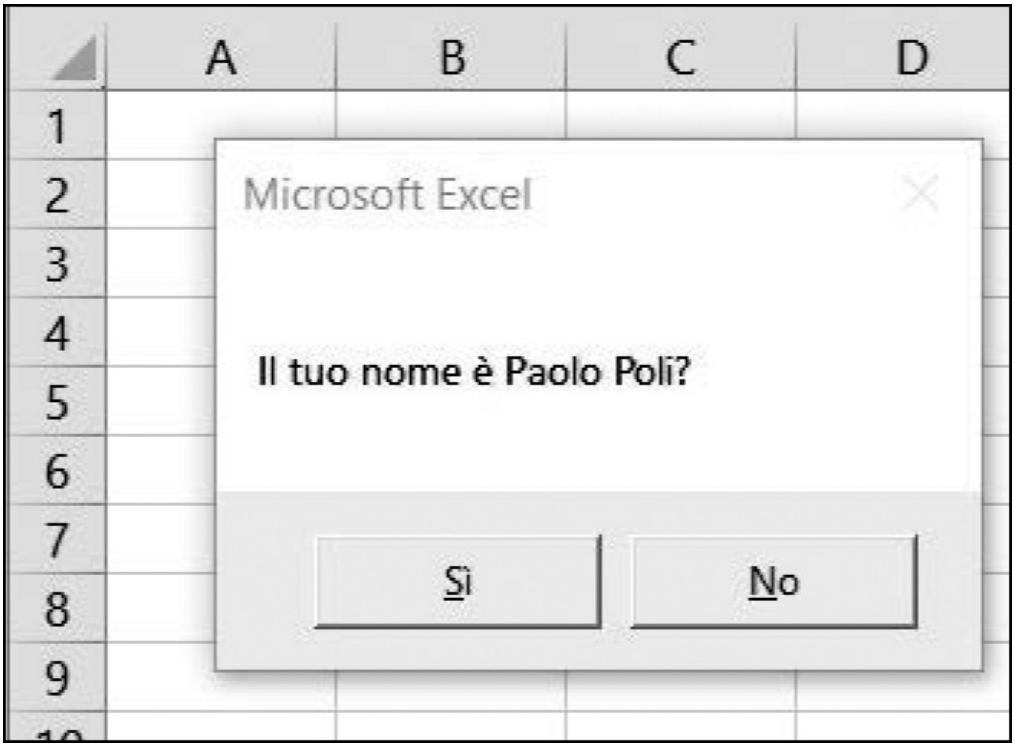

FIGURA 3.4 La routine GuessName visualizza questa finestra di dialogo.

ERRORE DI COMPILAZIONE?

C'è una possibilità che la macro GuessName non funzioni. Quando tentate di eseguirla, Excel potrebbe lamentarsi e visualizzare un messaggio di errore: `Errore di compilazione: variabile non definita`. Non preoccupatevi; esiste una soluzione semplice.

Se viene visualizzato questo errore, guardate la parte superiore del modulo e vedrete questo testo: `Option Explicit`. Basta eliminare quella riga e la macro dovrebbe funzionare. Quando nella parte superiore di un modulo è presente tale indicazione, significa che dovete "dichiarare" tutte le variabili (per ulteriori informazioni sulle variabili, vedete il Capitolo 7). Se è stata aggiunta questa riga, significa che il vostro VBE è impostato per aggiungerla automaticamente. Per ora, non preoccupatevi della cosa. Basta cancellarla e dimenticare la brusca interruzione.

Quando inserite il codice elencato nel passaggio 5, potreste notare che il VBE apporta alcune modifiche al testo che inserite. Per esempio, dopo aver digitato l'istruzione Sub, VBE inserisce automaticamente l'istruzione End Sub. E se omettete lo spazio prima o dopo un segno di uguale, VBE inserirà lo spazio per voi. Inoltre, il VBE cambia il colore e mette in maiuscole alcune parti del testo. Questo è perfettamente normale. È solo il modo in cui VBE mantiene le cose a posto e ben leggibili.

Se avete seguito i passaggi precedenti, avete appena scritto una routine Sub, ovvero una *macro*. Quando premete F5,

Excel esegue il codice e segue le istruzioni. In altre parole, Excel valuta ogni dichiarazione e fa ciò che gli avete chiesto di fare (ora… non è che per questa piccola cosa vi dobbiate montare la testa). Potete eseguire questa macro tutte le volte che volete, anche se, dopo una decina di volte, ho il sospetto che perderà molto del suo fascino.

Per la cronaca, questa semplice macro utilizza i seguenti concetti, che saranno trattati più avanti nel libro.

- Definire una routine Sub (la prima riga).
- Assegnare dei valori alle variabili (Msg e Ans).
- *Concatenare* (unire) più stringhe (usando l'operatore &).
- Utilizzare una funzione VBA (MsgBox).
- Utilizzare delle costanti VBA (vbYesNo, vbNo e vbYes).
- Utilizzare un costrutto If-Then (due volte).
- Terminare una routine Sub (l'ultima riga).

Non male per un principiante, vero?

Usare il registratore di macro

Un altro modo per ottenere il codice in un modulo VBA è registrare le vostre azioni, utilizzando il registratore di macro di Excel. Se avete svolto l'esercizio pratico presentato nel Capitolo 2, conoscete già questa tecnica.

RICORDA!

A proposito: non è assolutamente possibile registrare la routine GuessName mostrata nel paragrafo precedente. Potete registrare solo azioni che si possono effettivamente svolgere in Excel. La visualizzazione di una finestra di messaggi non è nelle normali prerogative di Excel (è una competenza di VBA). Il registratore di macro è utile, ma in molti casi probabilmente dovrete inserire almeno un po' di codice manualmente.

Ecco un esempio dettagliato che mostra come registrare una macro che crea un nuovo foglio di lavoro e nasconde tutte le righe tranne le prime dieci e tutte le colonne tranne le prime dieci. Se volete provare questo esempio, iniziate con una nuova cartella di lavoro vuota e seguite questi passaggi.

1. **Attivate un foglio di lavoro nella cartella di lavoro.**

   Qualsiasi foglio di lavoro andrà bene.

2. **Fate clic sulla scheda Sviluppo e assicuratevi che l'opzione Usa riferimenti relativi *non* sia evidenziata.**

   Questa macro è registrata usando riferimenti assoluti.

3. **Selezionate il comando Sviluppo ➪ Codice ➪ Registra macro oppure fate clic sull'icona accanto all'indicatore Pronto all'estremità sinistra della barra di stato.**

   Excel visualizza la finestra di dialogo Registra macro.

4. **Nella finestra di dialogo Registra macro, denominate la macro TenByTen, specificate che volete la macro memorizzata in questa cartella di lavoro e specificate Maiusc + T come tasto di scelta rapida.**

   La macro può essere eseguita premendo Ctrl + Maiusc + T.

5. **Fate clic su OK per avviare la registrazione.**

   Excel inserisce automaticamente un nuovo modulo VBA nel progetto che corrisponde alla cartella di lavoro attiva. Da questo momento in poi, Excel converte le vostre azioni in codice VBA. Mentre state registrando, l'icona nella barra di stato si trasforma in un quadratino. Questo è un promemoria che indica che il registratore di macro è in esecuzione. Inoltre potete fare clic su quell'icona per interrompere il registratore di macro.

6. **Fate clic sull'icona Nuovo foglio a destra dell'ultima scheda del foglio.**

   Excel inserirà un nuovo foglio di lavoro.

7. **Selezionate l'intera colonna K (l'undicesima) e premete Ctrl + Maiusc + freccia destra; quindi fate clic destro su una cella selezionata e selezionate dal menu contestuale il comando Nascondi.**

   Excel nasconderà tutte le colonne selezionate.

8. **Selezionate l'intera riga 11 e premete Ctrl + Maiusc + freccia giù; quindi fate clic destro su una cella selezionata e selezionate dal menu contestuale il comando Nascondi.**

   Excel nasconderà tutte le colonne selezionate.

9. **Selezionate la cella A1.**

10. **Selezionate il comando Sviluppo ➪ Codice ➪ Interrompi registrazione o fate clic sul pulsante Interrompi registrazione sulla barra di stato (il quadratino).**

Excel interromperà la registrazione.

Per visualizzare la composizione della macro che avete appena registrato, premete Alt + F11 per attivare il VBE. Individuate il nome della cartella di lavoro nel riquadro Progetto. Potete vedere che il progetto elenca un nuovo modulo. Il nome del modulo dipende dalla presenza di altri moduli nella cartella di lavoro quando avete avviato la registrazione della macro. In caso contrario, il modulo è denominato Modulo1. Fate doppio clic sul modulo e visualizzerete la finestra Codice per il modulo.

Ecco il codice generato:

```
Sub TenByTen()
'
' TenByTen Macro
'
' Keyboard Shortcut: Ctrl+Shift+T
'
    Sheets.Add After:=ActiveSheet
    Columns("K:K").Select
    Range(Selection, Selection.End(xlToRight)).Select
    Selection.EntireColumn.Hidden = True
    Rows("11:11").Select
    Range(Selection, Selection.End(xlDown)).Select
    Selection.EntireRow.Hidden = True
    Range("A1").Select
End Sub
```

Per provare questa macro, attivate un foglio di lavoro qualsiasi e premete il tasto di scelta rapida che le avete assegnato nel Punto 4: Ctrl + Maiusc + T.

Se non avete assegnato alcun tasto di scelta rapida, non preoccupatevi. Ecco come visualizzare un elenco di tutte le macro disponibili ed eseguire quella che desiderate.

1. **Selezionate il comando Sviluppo ➪ Codice ➪ Macro.**

   Se siete fan della tastiera potete premere Alt + F8. Uno qualsiasi di questi metodi visualizza la finestra di dialogo che elenca tutte le macro disponibili.

2. **Selezionate nell'elenco la macro (in questo caso è TenByTen).**
3. **Fate clic sul pulsante Esegui.**

   Excel eseguirà la macro e produrrà un nuovo foglio di lavoro con dieci righe e dieci colonne.

Mentre il registratore di macro è in esecuzione potete eseguire un numero qualsiasi di comandi e un numero qualsiasi di azioni. Excel convertirà diligentemente le azioni del mouse e le sequenze di tasti in codice VBA.

E, naturalmente, potete anche modificare la macro dopo averla registrata. Per mettere alla prova le vostre nuove abilità, provate a modificare la macro in modo che crei un foglio di lavoro di nove righe e colonne, perfetto per il Sudoku.

Copiare il codice VBA

L'ultimo metodo per inserire codice in un modulo VBA consiste nel copiarlo da un altro modulo o da un'altra posizione (di solito un sito web). Per esempio, una routine Sub o Function scritta per un progetto potrebbe essere utile anche in un altro. Invece di perdere tempo a reinserire il codice, potete attivare il modulo e utilizzare le normali procedure di copia-e-incolla degli Appunti (probabilmente siete piuttosto affezionati alle combinazioni di tasti Ctrl + C per copiare e Ctrl + V per incollare.) Dopo aver incollato il codice in un modulo VBA, se necessario, lo potrete modificare a piacere.

A questo proposito, troverete molti esempi di codice VBA sul Web. Se desiderate provarli, selezionate il codice nel browser e premete Ctrl + C per copiarlo. Quindi attivate un modulo e premete Ctrl + V per incollarlo.

Quando si copia del codice da un sito web, a volte è necessaria qualche modifica. Per esempio, le virgolette possono essere "tipografiche" e devono essere convertite in semplici virgolette. E a volte, le righe lunghe vanno a capo malamente. Le istruzioni errate sono facili da individuare in VBE, perché appaiono in rosso.

Personalizzare l'ambiente VBA

Se siete seriamente interessati a diventare programmatori Excel, trascorrerete molto tempo con i moduli VBA davanti agli occhi. Per aiutare a rendere le cose il più comode possibile (no, per favore non toglietevi le scarpe), VBE offre alcune opzioni di personalizzazione.

Quando è attivo il VBE, selezionate il comando Strumenti ➪ Opzioni. Verrà visualizzata una finestra di dialogo con quattro schede: Editor, Formato editor, Generale e Ancoraggio. Alcune delle loro opzioni più utili sono l'argomento dei prossimi paragrafi.

Utilizzare la scheda Editor

La Figura 3.5 mostra le opzioni alle quali potete accedere facendo clic sulla scheda Editor della finestra di dialogo

Opzioni. Potete usare le opzioni nella scheda Editor per controllare il funzionamento di determinate azioni in VBE.

FIGURA 3.5 La scheda Editor della finestra di dialogo Opzioni.

L'opzione Controllo automatico sintassi

L'impostazione Controllo automatico sintassi determina se il VBE visualizza una finestra di dialogo se rileva un errore di sintassi durante l'inserimento del codice VBA. La finestra di dialogo indica approssimativamente qual è il problema. Se non attivate questa impostazione, il VBE contrassegna gli errori di sintassi visualizzandoli in un colore diverso dal resto del codice e non avrete a che fare con alcuna finestra di dialogo.

L'opzione Dichiarazione di variabili obbligatoria

Se è attiva l'opzione Dichiarazione di variabili obbligatoria, VBE inserisce la seguente istruzione all'inizio di ogni nuovo modulo VBA inserito:

```
Option Explicit
```

La modifica di questa impostazione riguarda solo i nuovi moduli, non i moduli esistenti. Se nel vostro modulo appare questa istruzione, dovete definire esplicitamente ogni variabile che usate. Il Capitolo 7 entra nei dettagli del perché si dovrebbe avere questa abitudine.

L'opzione Elenco membri automatico

Se è attiva l'opzione Elenco membri automatico, VBE vi fornisce un aiuto quando inserisce del codice VBA. Mostra infatti un elenco che completerebbe logicamente l'istruzione che state digitando. Questa "magia" è chiamata *IntelliSense*.

Questa è una delle migliori caratteristiche di VBE. La Figura 3.6 mostra un esempio (che avrà molto più senso quando inizierete a scrivere codice VBA).

FIGURA 3.6 Un esempio di Elenco membri automatico.

L'opzione Informazioni rapide automatiche

Se è attiva l'opzione Informazioni rapide automatiche, durante la digitazione VBE visualizza le informazioni sulle funzioni e i relativi argomenti. Questo può essere veramente comodo. La Figura 3.7 mostra questa funzione in azione, mentre descrive gli argomenti per la funzione MsgBox.

FIGURA 3.7 Informazioni rapide automatiche sulla funzione MsgBox.

L'opzione Descrizione dati automatica

Se è attiva l'opzione Descrizione dati automatica, VBE visualizza il valore della variabile su cui è posizionato il cursore quando eseguite il debug del codice. Quando entrerete nel meraviglioso mondo del debugging, come descritto nel Capitolo 13, apprezzerete molto questa opzione.

L'impostazione Rientro automatico

L'impostazione Rientro automatico fa in modo che VBE indenti automaticamente ogni nuova riga di codice allo stesso modo della riga precedente.

CONSIGLIO

Per indentare il vostro codice usate il tasto Tab e non la barra spaziatrice. Inoltre, potete premere Maiusc + Tab per "de-indentare" una riga di codice. Se volete indentare più di una riga, prima selezionate tutte le righe che volete far rientrare e poi premete il tasto Tab.

CONSIGLIO

La barra degli strumenti Modifica di VBE (che è nascosta per impostazione predefinita) contiene due pulsanti utili:

Aumenta rientro e Riduci rientro. Questi pulsanti consentono di indentare o de-indentare rapidamente un blocco di codice. Selezionate il codice e fate clic su uno di questi pulsanti per cambiare il livello di rientro del blocco.

L'opzione Trascinamento della selezione

L'opzione Trascinamento della selezione, se abilitata, vi consente di copiare e spostare il testo trascinandolo e rilasciandolo con il mouse.

L'opzione Visualizza modulo intero

L'opzione Visualizza modulo intero imposta lo stato predefinito per i nuovi moduli (non influisce sui moduli esistenti). Se questa opzione è impostata, le routine nella finestra Codice appaiono come un unico elenco scorrevole. Se questa opzione è disattivata, potete visualizzare solo una routine alla volta.

L'opzione Separatore routine

Quando l'opzione Separatore routine è attivata, vengono visualizzate delle barre separatrici tra le routine presenti nella finestra Codice.

Utilizzare la scheda Formato editor

La Figura 3.8 mostra la scheda Formato editor della finestra di dialogo Opzioni. Con questa scheda, potete personalizzare l'aspetto di VBE.

FIGURA 3.8 Con la scheda Formato editor modificate l'aspetto di VBE.

L'opzione Colori codice

L'opzione Colori codice consente di impostare il colore del testo e il colore di sfondo dei vari elementi del codice VBA. Questa è solo una questione di preferenza personale.

L'opzione Tipo di carattere

L'opzione Tipo di carattere vi consente di selezionare il carattere usato nei moduli VBA. Per ottenere i migliori risultati, utilizzate un tipo di carattere a larghezza fissa, come Courier New. In un *font a larghezza fissa*, tutti i caratteri hanno esattamente la `stessa larghezza`. Ciò rende il vostro codice più leggibile, perché i caratteri sono allineati verticalmente, e potete facilmente distinguere i doppi spazi (che a volte è utile).

L'opzione Dimensione

L'impostazione Dimensione specifica le dimensioni in punti del carattere nei moduli VBA. Questa impostazione è una questione di preferenze personali, che dipende dalla risoluzione del vostro monitor e dal numero di carote che avete mangiato.

L'opzione Barra indicatori

Questa opzione controlla la visualizzazione della barra indicatori verticale al margine sinistro dei moduli. Dovreste tenerla attiva, altrimenti non sarete in grado di vedere gli utili indicatori grafici quando eseguirete il debug del codice.

Usare la scheda Generale

La Figura 3.9 mostra le opzioni disponibili nella scheda Generale della finestra di dialogo Opzioni. In quasi tutti i casi, le impostazioni predefinite sono perfette così come sono: non toccatele, almeno all'inizio.

FIGURA 3.9 La scheda Generale della finestra di dialogo Opzioni.

L'impostazione più importante è Intercettazione degli errori. È considerata una buona pratica utilizzare l'impostazione Interrompi ad ogni errore non gestito (che è l'impostazione predefinita). Se utilizzate un'altra impostazione, il codice di gestione degli errori non funzionerà. Per ulteriori informazioni al riguardo, consultate il Capitolo 12.

Se siete veramente interessati a queste opzioni, fate clic sul pulsante della Guida (?) per ulteriori dettagli.

Usare la scheda Ancoraggio

La Figura 3.10 mostra la scheda Ancoraggio. Queste opzioni determinano come si comportano le varie finestre in VBE. Quando una finestra è ancorata, viene fissata in posizione lungo uno dei bordi della finestra del programma VBE. Questo rende più facile l'identificazione e la localizzazione di una particolare finestra. Se si disattiva tutto l'ancoraggio, potreste aver aperte anche moltissime finestre. In genere, le impostazioni predefinite funzionano correttamente.

FIGURA 3.10 La scheda Ancoraggio della finestra di dialogo Opzioni.

RICORDA!

A volte il VBE sembra avere una propria precisa opinione quando si cerca di agganciare una finestra. Se l'aggancio non vi sembra funzionare correttamente, cercate di capire che cosa preferisce e accontentatelo.

Capitolo 4

Introduzione al modello a oggetti di Excel

IN QUESTO CAPITOLO

- **Introduzione al concetto di oggetti**
- **La gerarchia degli oggetti di Excel**
- **Collezioni di oggetti**
- **Riferimenti a oggetti specifici nel codice VBA**
- **Lettura o modifica delle proprietà di un oggetto**
- **Esecuzione di azioni con i metodi di un oggetto**

Tutti hanno familiarità con la parola *oggetto*. Bene, dimenticate la definizione che avete in mente. Nel mondo della programmazione, la parola *oggetto* ha un significato specifico. Viene spesso usata come parte dell'espressione *programmazione orientata agli oggetti* o *programmazione a oggetti*, per semplicità *OOP*. L'OOP si basa sull'idea che il software opera su oggetti che hanno attributi (proprietà) e che possono essere manipolati. Questi oggetti non sono cose materiali: esistono sotto forma di bit e byte.

In questo capitolo introduco il modello a oggetti di Excel, che è una struttura gerarchica di oggetti, contenuti in Excel. Al termine del capitolo, vi sarete fatti un'idea di cosa sia l'OOP e del perché sia necessario comprendere questo concetto per diventare programmatori VBA. In fin dei conti, la programmazione di Excel si riduce davvero alla manipolazione di oggetti. Non è difficile.

Excel è un oggetto?

Avete sicuramente usato Excel, ma probabilmente non lo avete mai considerato un oggetto. Più lavorerete con VBA, più penserete a Excel in questi termini. Capirete che Excel è un oggetto e che contiene altri oggetti. Questi oggetti, a loro volta, contengono altri oggetti. In altre parole, la programmazione VBA opera su una gerarchia di oggetti.

Al vertice di questa gerarchia si trova l'oggetto Application, in questo caso l'applicazione Excel (la madre di tutti gli oggetti).

Muoversi nella gerarchia di oggetti

L'oggetto Application contiene altri oggetti. Di seguito sono riportati alcuni degli oggetti più utili contenuti nell'applicazione Excel:

- Addin;
- Window;
- Workbook;
- WorksheetFunction.

Ogni oggetto contenuto nell'oggetto Application può contenere altri oggetti. Per esempio, di seguito sono riportati alcuni oggetti che possono essere contenuti in un oggetto Workbook:

- Chart (che è un foglio grafico);
- Name;
- VBProject;
- Window;

» Worksheet.

A loro volta, ciascuno di questi oggetti può contenere ancora altri oggetti. Consideriamo un oggetto Worksheet, il quale è contenuto in un oggetto Workbook, che è contenuto nell'oggetto Application. Alcuni degli oggetti che possono essere contenuti in un oggetto Worksheet sono:

» Comment;
» Hyperlink;
» Name;
» PageSetup;
» PivotTable;
» Range.

In altre parole, se volete fare qualcosa con un intervallo (Range) su un particolare foglio di lavoro (Worksheet), potrebbe esservi utile immaginare tale intervallo nel seguente modo:

Intervallo (Range) ➪ contenuto nel foglio di lavoro (Worksheet) ➪ contenuto nella cartella di lavoro (Workbook) ➪ contenuto in Excel (Application)

La cosa inizia ad avere un senso?

RICORDA!

Quando inizierete le vostre ricerche, scoprirete che Excel ha ben più oggetti di quanti ne possiate immaginare. Anche gli utenti più esperti possono esserne sopraffatti. La buona notizia è che della maggior parte di questi oggetti non dovrete mai occuparvene. Quando lavorate su un problema, potete concentrarvi solo su alcuni specifici oggetti, che potrete scoprire registrando una macro.

Le collezioni

Le collezioni sono un altro concetto chiave nella programmazione VBA. Una *collezione* è un gruppo di oggetti dello stesso tipo. E per complicare le cose, una collezione è anch'essa un oggetto.

Ecco alcuni esempi di collezioni comunemente utilizzate.

» **Workbooks:** una collezione di tutti gli oggetti cartella di lavoro (Workbook) attualmente aperti.
» **Worksheets:** una collezione di tutti gli oggetti foglio di lavoro (Worksheet) contenuti in un particolare oggetto cartella di lavoro (Workbook).
» **Charts:** una collezione di tutti gli oggetti foglio grafico (Chart) contenuti in un particolare oggetto cartella di lavoro (Workbook).
» **Sheets:** una collezione di tutti i fogli (indipendentemente dal loro tipo) contenuti in un particolare oggetto cartella di lavoro (Workbook).

Potete notare che i nomi delle collezioni sono tutti al plurale: la cosa ha senso.

"A cosa servono le collezioni?", potreste giustamente chiedere. Sono molto utili, per esempio, quando non volete lavorare con un solo foglio di lavoro, ma con un paio di questi o con tutti. Come vedrete, il codice VBA può scorrere tutti i membri di una collezione e fare qualcosa su ciascuno di essi.

Far riferimento agli oggetti

Far riferimento a un oggetto è importante, perché permette di identificare l'oggetto con il quale si vuole lavorare. Dopotutto, VBA non può leggere nella vostra mente. Non ancora, almeno.

Potete lavorare su un'intera collezione di oggetti in una sola volta. Più spesso, tuttavia, è necessario lavorare con un oggetto specifico di una collezione (per esempio uno specifico foglio di lavoro in una cartella di lavoro). Per far riferimento a un singolo oggetto di una collezione, dovete indicare, dopo il nome della collezione, il nome dell'oggetto o il numero di indice tra parentesi, in questo modo:

```
Worksheets("Foglio1")
```

Notate che il nome del foglio è racchiuso tra virgolette. Se omettete le virgolette, Excel non sarà in grado di identificare l'oggetto (e supporrà che si tratti del nome di una variabile).

Se desiderate lavorare con il primo foglio di lavoro nella collezione, potete anche utilizzare il seguente riferimento:

```
Worksheets(1)
```

RICORDA!

In questo caso, il numero non è tra virgolette. La morale? Se fate riferimento a un oggetto usando il suo nome, inserite le virgolette. Se fate riferimento a un oggetto utilizzando il suo indice, utilizzate un numero semplice senza virgolette.

Che dire dei fogli grafici? Un foglio grafico contiene un singolo grafico. Ha una scheda, ma non è un foglio di lavoro. Bene, come è logico, il modello a oggetti ha una collezione chiamata Charts. Questa collezione contiene tutti gli oggetti foglio grafico in una cartella di lavoro (mentre non include i grafici incorporati in un foglio di lavoro). E per mantenere la logica delle cose, c'è un'altra collezione chiamata Sheets, la quale contiene tutti i fogli (di lavoro e grafici) di una cartella di lavoro. La collezione Sheets è utile se desiderate lavorare con tutti i fogli di una cartella di lavoro, indipendentemente dal fatto che si tratti di fogli di lavoro o fogli grafici.

Quindi, un singolo foglio di lavoro denominato Foglio1 è un membro di due collezioni: la collezione Worksheets e la collezione Sheets. Quindi potete farvi riferimento in due modi:

```
Worksheets("Foglio1")
Sheets("Foglio1")
```

Navigare nella gerarchia

Se desiderate lavorare con gli oggetti di Excel, li troverete tutti sotto l'oggetto Application. Quindi iniziate digitando **Application**.

Ogni altro oggetto nel modello a oggetti di Excel si trova sotto l'oggetto Application. Si arriva a questi oggetti spostandosi lungo la gerarchia e usando come separatore il punto (.). Per accedere all'oggetto Workbook denominato `Cartel1.xlsx`, partite dall'oggetto Application e scendete fino all'oggetto collezione Workbooks:

```
Application.Workbooks("Cartel1.xlsx")
```

Per accedere a un determinato foglio di lavoro, aggiungete un operatore punto e accedete all'oggetto collezione Worksheets:

```
Application.Workbooks("Cartel1.xlsx").Worksheets(1)
```

Non è sufficiente? Se desiderate ottenere il valore dalla cella `A1` del primo foglio di lavoro della cartella di lavoro denominata `Cartel1.xlsx`, dovrete scendere di un altro livello, sull'oggetto Range:

```
Application.Workbooks("Cartel1.xlsx").Worksheets(1).
Range("A1").Value
```

Quando fate riferimento a un oggetto Range in questo modo, si parla di *riferimento completo*. Avete indicato a Excel esattamente quale intervallo desiderate, in quale foglio di lavoro e in quale cartella di lavoro, e non avete lasciato nulla all'immaginazione. L'immaginazione va bene per le persone, ma non per i programmi per computer.

Comunque, i nomi delle cartelle di lavoro hanno anche un punto per separare il nome del file dall'estensione (per esempio, `Cartel1.xlsx`). Questa è solo una coincidenza. Il punto in un nome di file non ha nulla a che fare con l'operatore punto a cui abbiamo fatto riferimento pochi paragrafi fa.

Semplificare i riferimenti agli oggetti

Se doveste qualificare completamente ogni riferimento a un oggetto creato, il vostro codice diventerebbe piuttosto lungo e anche difficile da leggere. Fortunatamente, Excel offre alcune scorciatoie che possono migliorare la leggibilità (e farvi risparmiare la digitazione). Per iniziare, l'oggetto Application è sempre dato per scontato. Ci sono solo pochi casi in cui ha senso scriverlo. L'omissione del riferimento all'oggetto Application riduce l'esempio del paragrafo precedente a:

```
Workbooks("Cartel1.xlsx").Worksheets(1).Range("A1").Value
```

Questo è già qualcosa. Ma aspettate, c'è di più. Se siete sicuri che `Cartel1.xlsx` sia la cartella di lavoro attiva, potete anche omettere quel riferimento. Così potete scrivere:

```
Worksheets(1).Range("A1").Value
```

Ora va meglio. Avete indovinato la prossima scorciatoia? Esatto. Se sapete che il primo foglio di lavoro è il foglio di

lavoro attualmente attivo, Excel lo darà per scontato e vi consentirà di digitare semplicemente:

```
Range("A1").Value
```

RICORDA!

Contrariamente a quanto possiate pensare, Excel non ha un oggetto Cell. Una *cella* è semplicemente un oggetto Range costituito da un solo elemento.

Le scorciatoie descritte qui sono ottime, ma possono anche essere pericolose. Che cosa succede se *pensate* (erroneamente) che `Cartel1.xlsx` sia la cartella di lavoro attiva? Potreste ricevere un errore o, peggio, potreste ottenere il valore sbagliato e non vi rendereste nemmeno conto dell'errore. Per questo motivo, spesso è meglio specificare completamente i riferimenti agli oggetti.

Il Capitolo 14 descrive la struttura With-End With, che aiuta a specificare completamente i riferimenti, ma aiuta anche a rendere più leggibile il codice e a ridurre la digitazione. Il meglio del meglio!

Le proprietà e i metodi

Sebbene sia importante sapere come far riferimento agli oggetti, non potete fare nulla di utile facendo semplicemente riferimento a un oggetto (come negli esempi nei paragrafi precedenti). Per realizzare qualcosa di significativo, occorre fare una delle due cose:

- leggere o modificare le *proprietà* di un oggetto;
- specificare un *metodo*, un'azione da applicare a un oggetto.

PER I PIÙ CURIOSI

UN'ANALOGIA CULINARIA

Ecco un'analogia: paragoniamo Excel a una catena di fast-food, cosa che può aiutarvi a comprendere meglio le relazioni esistenti tra oggetti, proprietà e metodi in VBA.

L'unità di base di Excel è un oggetto Workbook. In una catena di fast food, l'unità di base è il singolo ristorante. In Excel, potete aprire una nuova cartella di lavoro (Workbook) o chiudere una cartella di lavoro e tutte le cartelle di lavoro aperte sono note come Workbooks (la collezione di tutti gli oggetti Workbook). Allo stesso modo, la gestione di una catena di fast-food può aggiungere un ristorante e chiudere un ristorante, e tutti i ristoranti della catena possono essere considerati come la collezione dei ristoranti (una collezione di oggetti ristorante).

Una cartella di lavoro di Excel è un oggetto in sé, ma contiene anche altri oggetti, come fogli di lavoro, fogli grafici, moduli VBA e così via. Inoltre, ogni oggetto di una cartella di lavoro può contenere i propri oggetti. Per esempio, un oggetto Worksheet può contenere oggetti Range, oggetti PivotTable, oggetti Shape e così via.

Continuando con l'analogia, un oggetto Ristorante (come una cartella di lavoro) contiene oggetti come Cucina, AreaClienti e Tavoli (questa è una collezione). Inoltre, la gestione può aggiungere o rimuovere oggetti dall'oggetto Ristorante. Per esempio, può aggiungere più oggetti Tavolo alla collezione Tavoli. Ognuno di questi oggetti può contenere altri oggetti. Per esempio, l'oggetto Cucina ha un oggetto PianoCottura, l'oggetto CappaAspirante, l'oggetto Chef, l'oggetto Lavandino e così via.

Fin qui tutto bene. L'analogia sembra funzionare.

Gli oggetti di Excel hanno anche proprietà. Per esempio, un oggetto Range ha proprietà come Value e Name e un oggetto Shape ha proprietà come Width, Height e così via. Non sorprendemente anche gli oggetti di un fast-food hanno proprietà. L'oggetto PianoCottura, per esempio, ha proprietà come Temperatura e NumeroFornelli. CappaAspirante ha il proprio insieme di proprietà (Acceso, Velocità e così via).

Oltre alle proprietà, gli oggetti di Excel hanno anche dei metodi che eseguono un'operazione su un oggetto. Per esempio, il metodo ClearContents cancella il contenuto di un oggetto Range. Anche un oggetto in un fast-food ha dei metodi. Potete facilmente immaginare un metodo RegolaTermostato per un oggetto PianoCottura o un metodo Accendi per un oggetto CappaAspirante.

In Excel, i metodi a volte modificano le proprietà di un oggetto. Il metodo ClearContents modifica la proprietà Value di un Range. Allo stesso modo, il metodo RegolaTermostato di un oggetto PianoCottura influisce sulla sua proprietà Temperatura. Con VBA, potete scrivere routine per manipolare gli oggetti di Excel. In un fast-food, la direzione può dare ordini di manipolare gli oggetti nei ristoranti ("Accendi il piano cottura e accendi la cappa aspirante").

La prossima volta che visitate il vostro fast food preferito, dite semplicemente "Usa il metodo Grill su un oggetto Burger con la proprietà Cipolle impostata su False".

Proprietà di un oggetto

Ogni oggetto ha delle proprietà. Le *proprietà* sono come attributi che descrivono l'oggetto. Le proprietà di un oggetto determinano il suo aspetto, come si comporta e anche se è visibile o meno. Usando VBA, potete fare due operazioni con le proprietà di un oggetto:

- » esaminare l'impostazione attuale di una proprietà;
- » modificare le impostazioni della proprietà.

Un oggetto Range di una sola cella, per esempio, ha una proprietà denominata Value. La proprietà Value memorizza il valore contenuto nella cella. Potete scrivere del codice VBA per visualizzare la proprietà Value oppure scrivere del codice VBA per impostare la proprietà Value con un valore specifico. La seguente macro utilizza la funzione MsgBox, incorporata in VBA, per visualizzare una finestra che mostra il valore contenuto nella cella A1 del Foglio1 della cartella di lavoro attiva (vedi Figura 4.1):

```
Sub ShowValue()
     Contents = Worksheets("Foglio1").Range("A1").Value
     MsgBox Contents
End Sub
```

Tra l'altro, MsgBox è una funzione molto utile. Potete usarla per visualizzare i risultati mentre Excel esegue il vostro codice VBA. Nel Capitolo 15 scoprirete di più su questa funzione, quindi siate pazienti (oppure fateci un salto adesso e leggetelo tutto).

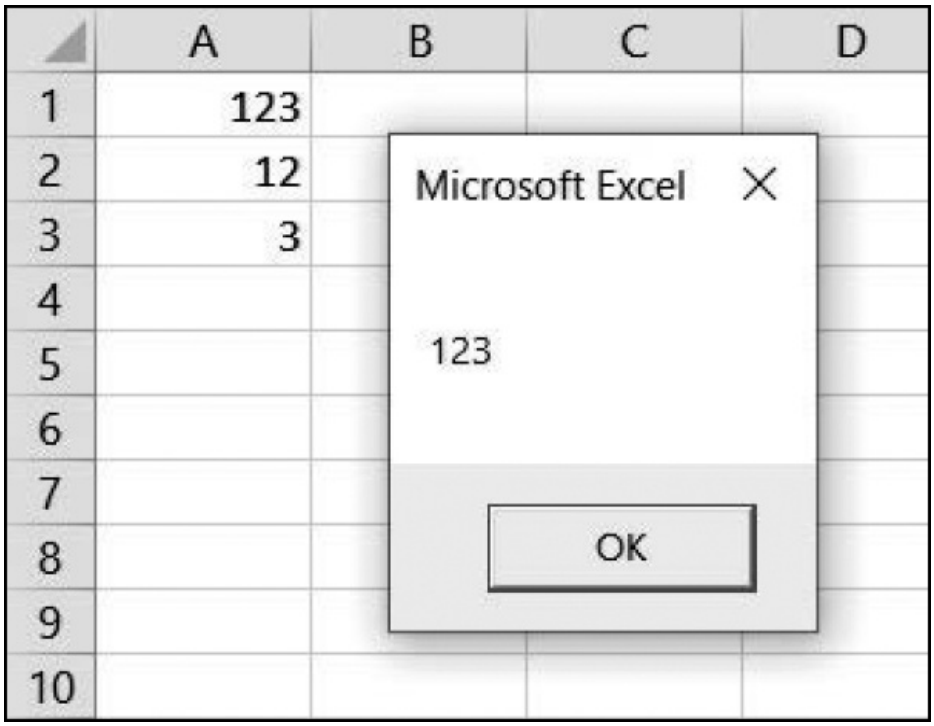

FIGURA 4.1 Questa finestra visualizza la proprietà Value dell'oggetto Range.

Il codice nell'esempio precedente mostra l'impostazione corrente della proprietà Value di una cella. Che cosa succede se desiderate modificare il valore di quella proprietà? La seguente macro modifica il valore nella cella A1 agendo sulla proprietà Value della cella stessa:

```
Sub ChangeValue()
     Worksheets("Foglio1").Range("A1").Value = 994.92
End Sub
```

Dopo che Excel avrà eseguito questa routine, la cella A1 del Foglio1 della cartella di lavoro corrente conterrà il valore 994,92. Se la cartella di lavoro attiva non ha un foglio denominato Foglio1, il risultato dell'esecuzione di quella macro sarà un messaggio di errore. VBA segue le istruzioni e non può funzionare con un foglio che non esiste.

WARNING

Il lettore attento avrà notato che il valore inserito nel codice conteneva un punto decimale (994.92), mentre nella cella A1 è comparso quello stesso numero ma con la virgola decimale (994,92). In pratica il linguaggio di programmazione VBA usa lo standard americano (il punto), ma le opzioni locali italiane di Excel interpretano e visualizzano quello stesso numero correttamente, secondo lo standard italiano. Il vantaggio: il codice che scriverete potrà essere impiegato senza interventi in tutto il mondo.

Ogni oggetto ha un proprio insieme di proprietà, sebbene alcune proprietà siano comuni a molti oggetti. Per esempio, molti oggetti (ma non tutti) hanno una proprietà Visible. La maggior parte degli oggetti ha anche una proprietà Name.

Alcune proprietà dell'oggetto sono di sola lettura; questo significa che il codice può ottenere il valore della proprietà, ma non può modificarlo. L'oggetto Application, per esempio, ha una proprietà denominata Version, la quale restituisce il numero di versione di Excel. Non potete modificare la proprietà Version: è di sola lettura.

RICORDA!

Come accennato in precedenza in questo capitolo, una collezione è anche un oggetto. Ciò significa che anche una collezione ha delle proprietà. Per esempio, per sapere quante cartelle di lavoro sono aperte potete accedere alla proprietà Count della collezione Workbooks. La seguente routine VBA visualizza una finestra che indica il numero di cartelle di lavoro aperte:

```
Sub CountBooks()
     MsgBox Workbooks.Count
End Sub
```

Metodi di un oggetto

Oltre alle proprietà, gli oggetti possiedono anche dei metodi. Un *metodo* è un'azione che viene eseguita su un oggetto. Un metodo può modificare le proprietà di un oggetto o far fare qualcosa all'oggetto.

Questo semplice esempio utilizza il metodo ClearContents su un oggetto Range per cancellare il contenuto di dodici celle del foglio attivo:

```
Sub ClearRange()
     Range("A1:A12").ClearContents
End Sub
```

Alcuni metodi accettano uno o più argomenti. Un *argomento* è un valore che specifica l'azione da eseguire. Dovete specificare gli argomenti di un metodo dopo il nome del metodo, separati da uno spazio. Se gli argomenti sono più d'uno, dovrete separarli con una virgola.

L'esempio seguente attiva il Foglio1 (nella cartella di lavoro attiva) e poi copia il contenuto della cella `A1` nella cella `B1` utilizzando il metodo Copy dell'oggetto Range. In questo esempio, il metodo Copy ha un argomento: l'intervallo di destinazione dell'operazione di copia:

```
Sub CopyOne()
     Worksheets("Foglio1").Activate
     Range("A1").Copy Range("B1")
End Sub
```

Notate che il riferimento al foglio di lavoro si omette quando fate riferimento a oggetti Range. Potete farlo in modo sicuro grazie all'istruzione che attiva il Foglio1 (utilizzando il metodo Activate).

Un altro modo per specificare un argomento per un metodo consiste nell'usare il nome dell'argomento seguito dalla sequenza due punti e uguale. L'utilizzo degli argomenti con nome è facoltativo, ma spesso il suo uso può rendere più comprensibile il codice. La seconda istruzione della routine CopyOne potrebbe essere scritta anche in questo modo:

```
Range("A1").Copy Destination:=Range("B1")
```

Nella Figura 4.2, notate il piccolo prompt mentre scrivete la frase. Quel prompt mostra il nome dell´argomento.

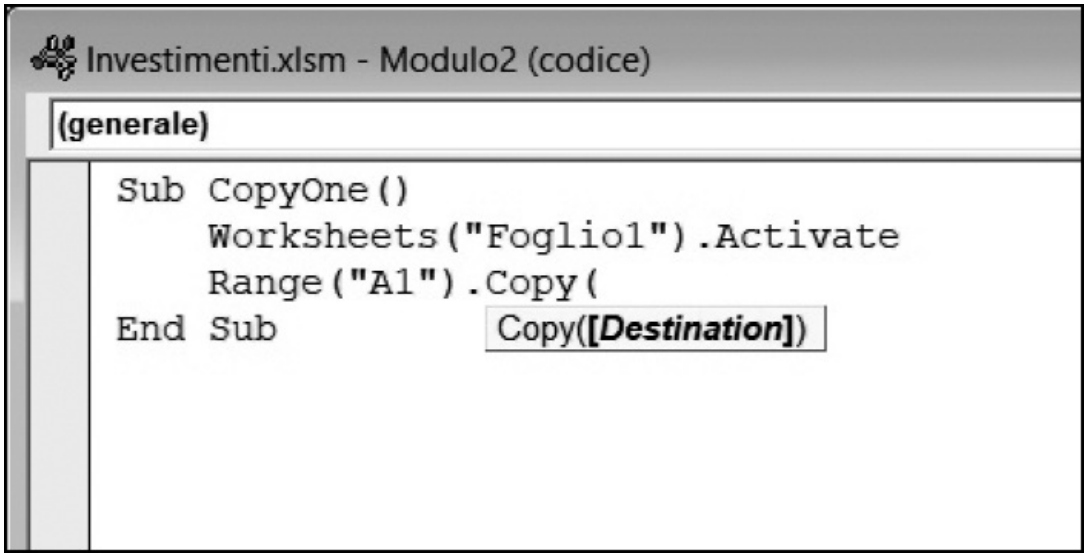

FIGURA 4.2 Mentre digitate, VBE visualizza un elenco di argomenti.

Poiché una collezione è anche un oggetto, anche le collezioni hanno dei metodi. La seguente macro utilizza il metodo Add della collezione Workbooks:

```
Sub AddAWorkbook ()
     Workbooks.Add
End Sub
```

Come potete immaginare, questa istruzione crea una nuova cartella di lavoro. In altre parole, *aggiunge* una nuova cartella di lavoro alla collezione Workbooks. Dopo aver eseguito questa macro, la nuova cartella di lavoro sarà anche attiva.

Eventi per gli oggetti

Questo paragrafo tocca solo brevemente un altro argomento che è necessario conoscere: gli eventi. Gli oggetti rispondono ai vari *eventi* che si verificano. Per esempio, quando si lavora in Excel e si attiva un'altra cartella di lavoro, si verifica un evento di attivazione di tale cartella di lavoro. Per esempio, potete avere una macro VBA progettata per essere eseguita ogni volta che si verifica un evento Activate su un determinato oggetto Workbook.

Excel supporta molti eventi, ma non tutti gli oggetti possono rispondere a tutti gli eventi. E alcuni oggetti non rispondono ad alcun evento. Gli unici eventi che potete utilizzare sono quelli resi disponibili dai programmatori di Microsoft Excel. Il concetto di evento diventerà più chiaro nel Capitolo 11 e anche nella Parte 4.

Scoprire di più

Ora siete ufficialmente entrati nel meraviglioso mondo degli oggetti, delle proprietà, dei metodi e degli eventi. Scoprirete di più su questi concetti nei capitoli che seguono. Se non siete abbastanza soddisfatti, potreste essere interessati anche ad altri tre eccellenti strumenti:

- la Guida di VBA;
- Il Visualizzatore oggetti;
- l'Elenco membri automatico.

Utilizzare la Guida di VBA

La Guida di VBA descrive ogni oggetto, proprietà e metodo disponibile e fornisce anche del codice di esempio. Questa è un'ottima risorsa per scoprire VBA ed è più completa di qualsiasi libro esistente sul mercato. Ma è anche molto noiosa da leggere. In più è solo in lingua inglese (almeno per il momento).

RICORDA!

Se utilizzate Excel 2013 o una versione successiva, dovrete essere connessi a Internet per utilizzare la Guida di VBA (per le versioni precedenti non esiste questo requisito). Tuttavia, potete scaricare la Guida di VBA dal sito web di Microsoft. Fate una ricerca sul Web per scaricare la documentazione di Excel VBA.

Se state lavorando in un modulo VBA e avete bisogno di informazioni su un determinato oggetto, metodo o proprietà, spostate il cursore sulla parola che vi interessa e premete F1. In pochi secondi, otterrete lo specifico argomento della Guida, visualizzato nel browser web, completo di riferimenti incrociati e spesso anche con un esempio o due.

La Figura 4.3 mostra parte di una schermata del sistema della Guida di VBA, in questo caso per un oggetto Worksheet.

Usare il Visualizzatore oggetti

VBE include un altro strumento: il Visualizzatore oggetti. Come suggerisce il nome, questo strumento consente di navigare tra gli oggetti disponibili. Per accedere al Visualizzatore oggetti, premete F2 quando è attivo il VBE (o selezionate Visualizza ➪ Visualizzatore oggetti). Verrà visualizata una finestra come quella rappresentata nella Figura 4.4.

FIGURA 4.3 Un esempio della Guida di VBA. Purtroppo è disponibile solo in lingua inglese.

FIGURA 4.4 Esplorazione degli oggetti con il Visualizzatore oggetti.

L'elenco a discesa in alto presenta tutte le librerie di oggetti attualmente disponibili. La Figura 4.4 mostra Tutte le librerie. Se desiderate esplorare gli oggetti di Excel, selezionare Excel dall'elenco a discesa.

Il secondo elenco a discesa è dove si inserisce una stringa di ricerca. Per esempio, se desiderate esaminare tutti gli oggetti di Excel che trattano i commenti, digitate **comment** (eh già, in inglese) nel secondo campo e fare clic sul pulsante Cerca in (quello con il binocolo). Il riquadro Risultati ricerca mostrerà tutto ciò che nella libreria di oggetti contiene il testo *comment*. Se notate qualcosa che potrebbe interessarvi, selezionatelo e premete F1 per ulteriori informazioni online.

Elenco automatico di proprietà e metodi

Il Capitolo 3 introduce una comoda funzionalità di nome Elenco membri automatico. Mentre digitate, questa funzione vi suggerisce un elenco di proprietà e metodi. La Figura 4.5 mostra un esempio per la collezione Workbooks.

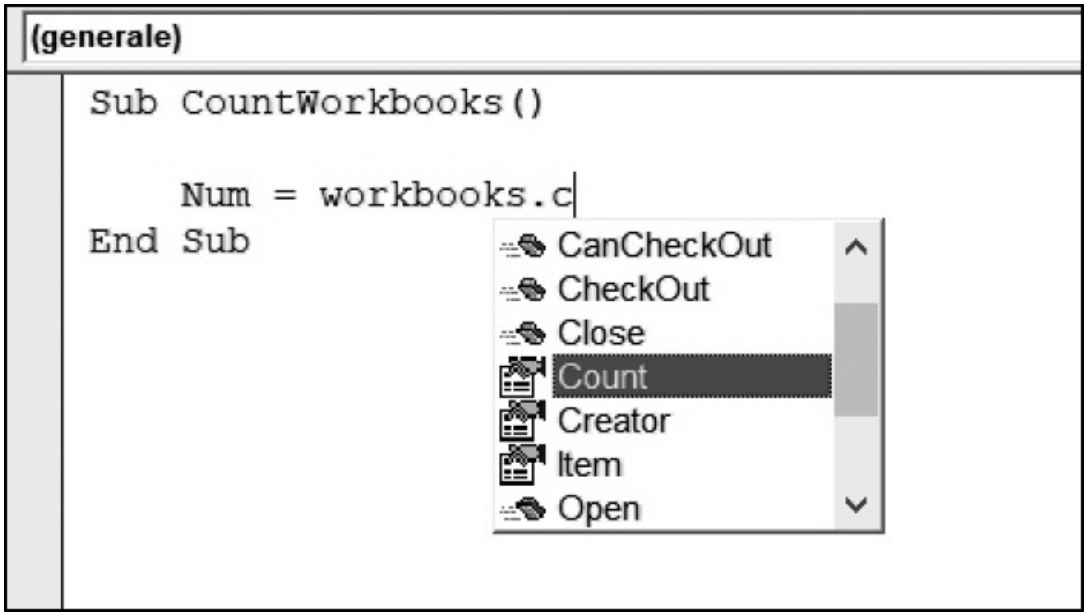

FIGURA 4.5 La funzione Elenco membri automatico consente di identificare proprietà e metodi di un oggetto.

Quando avrete inserito il punto dopo *workbooks*, VBE vi offre il suo aiuto visualizzando un elenco di proprietà e metodi per tale collezione. Digitando una lettera (per esempio la lettera *c*), l'elenco si riduce agli elementi che iniziano con quella lettera. A ogni lettera digitata, VBE restringe ulteriormente l'elenco delle scelte. Selezionate l'oggetto che vi serve, premete Tab e voilà! Avrete risparmiato alcune digitazioni e vi siete assicurati che la proprietà o il metodo sia stato digitato correttamente.

Capitolo 5

Routine Sub e Function in VBA

IN QUESTO CAPITOLO

- **Le differenze tra routine Sub e routine Function**
- **Esecuzione delle routine Sub (in molti modi)**
- **Esecuzione delle routine Function (in due modi)**

Nei capitoli precedenti, avete già visto il termine *routine Sub* e *routine Function*. Probabilmente le differenze tra i due tipi di routine sono per voi ancora un mistero, ma non temete. Questo capitolo chiarisce qualsiasi confusione su questi concetti.

Routine Sub e Function

Il codice VBA che scrivete in Visual Basic Editor forma delle *routine*. I due tipi più comuni di routine sono le routine Sub e le routine Function.

- Una *routine Sub* è costituita da un gruppo di istruzioni VBA che esegue un'azione (o una sequenza di azioni) con Excel.
- Una *routine Function* è un gruppo di istruzioni VBA che esegue un calcolo e restituisce un singolo valore (o, talvolta, un'intera serie di valori).

La maggior parte delle macro che scrivete in VBA è costituita da routine Sub. Potete considerare una routine Sub come un comando: eseguite la routine Sub e succede *qualcosa* (naturalmente, un qualcosa che dipende dal codice VBA contenuto nella routine Sub).

Anche una Function è una routine, ma è un po' diversa da una Sub. Conoscete già il concetto di funzione. Excel include molte funzioni nel foglio di lavoro che usate ogni giorno (beh, almeno ogni giorno lavorativo). Alcuni esempi sono SOMMA, RATA e CERCA.VERT. Queste funzioni del foglio di lavoro si utilizzano nelle formule. Ogni funzione accetta uno o più argomenti (sebbene alcune funzioni non utilizzino argomenti). La funzione esegue alcuni calcoli dietro le quinte usando quegli argomenti e infine restituisce un singolo valore. Le routine Function sviluppate con VBA funzionano allo stesso modo.

Le routine Sub

Ogni routine Sub inizia con la parola chiave Sub e termina con una dichiarazione End Sub. Ecco un esempio:

```
Sub ShowMessage()
     MsgBox "That's all folks!"
End Sub
```

Questo esempio mostra una routine chiamata ShowMessage. Dopo il nome della routine vi è una coppia di parentesi. Nella maggior parte dei casi, queste parentesi sono vuote. Tuttavia, potete anche passare degli argomenti da altre routine alle routine Sub. Se la Sub utilizza argomenti, elencateli tra parentesi.

RICORDA!

Quando registrate una macro con il registratore di macro di Excel, il risultato è sempre una routine Sub che non accetta argomenti.

Come vedrete più avanti in questo capitolo, Excel offre diversi modi per eseguire una routine Sub.

Le routine Function

Ogni routine Function inizia con la parola chiave Function e termina con un'istruzione End Function. Ecco un semplice esempio:

```
Function CubeRoot(number)
   CubeRoot = number ^ (1 / 3)
End Function
```

Questa funzione, CubeRoot, accetta un argomento (una variabile di nome *number*), racchiuso tra parentesi. Le funzioni possono avere fino a 255 argomenti o anche nessuno. Quando eseguite questa funzione, essa restituisce un singolo valore: la radice cubica dell'argomento che le avete passato.

CONSIGLIO

VBA consente di specificare quale tipo di informazioni (il *tipo di dati*) viene restituito da una routine Function. Per esempio, un valore può essere una valuta, una data o una stringa di testo. Il Capitolo 7 contiene ulteriori informazioni sulla specifica dei tipi di dati.

Potete eseguire una routine Function in due soli modi: da un'altra routine (una routine Sub o un'altra routine Function) o utilizzarla in una formula del foglio di lavoro.

RICORDA!

Per quanto possiate provare, non potete utilizzare il registratore di macro di Excel per registrare una routine Function. Dovrete inserire manualmente ogni routine Function che create.

Denominare le routine Sub e Function

Come per gli esseri umani, gli animali domestici e gli uragani, ogni routine Sub e Function deve avere un nome. Anche se è perfettamente lecito dare al proprio cane il nome Amilcare Palladipelo, di solito non è una buona idea adottare un atteggiamento così fantasioso quando si dà un nome alle routine. È necessario seguire alcune regole.

- Potete utilizzare lettere, numeri e alcuni caratteri di punteggiatura, ma il primo carattere deve essere una lettera.
- Non potete utilizzare spazi o punti nel nome.
- VBA non distingue tra lettere maiuscole e minuscole.
- Non potete utilizzare nessuno dei seguenti caratteri nel nome di una routine: #, $, %, &, @, ^, * o !. In altre parole, il nome della routine non può sembrare una parola dei fumetti.
- Se scrivete una routine Function per una formula, evitate di utilizzare un nome simile a un indirizzo di cella (per esempio, A1 o B52). In realtà, Excel consente di assegnare tali nomi alle funzioni, ma perché mai dovreste complicarvi le cose più di quanto già non lo siano?
- I nomi delle routine non possono essere più lunghi di 255 caratteri (sì, ma perché mai qualcuno sceglierebbe un nome di routine così lungo?).

Idealmente, il nome di una routine descrive lo scopo della routine stessa. Una buona prassi è quella di creare un nome combinando un verbo e un sostantivo, per esempio, ElaboraDati, StampaReport, Ordina_Matrice o Controlla_NomeFile.

A proposito: se dovete scrivere routine professionali, prediligete nomi in inglese. Le vostre routine risulteranno comprensibili in tutto il mondo.

Alcuni programmatori preferiscono impiegare nomi simili a frasi, che forniscono una descrizione completa della routine. Alcuni esempi potrebbero essere ScriviReportSuFileDiTesto e Chiedi_opzioni_di_stampa_e_poi_stampa_il_report. L'uso di nomi così lunghi ha dei pro e dei contro. Da un lato, tali nomi sono descrittivi e in genere non ambigui. D'altro canto, ci vuole parecchio tempo per digitarli. Ognuno sviluppa lo stile di denominazione che preferisce, ma se la macro non è solo di tipo "mordi e fuggi", è meglio che il suo nome sia descrittivo; evitate anche i nomi privi di significato, come Esegui, Aggiorna, Correggi e la sempre popolare Macro1.

Eseguire le routine Sub

Anche se a questo punto potreste non sapere molto sullo sviluppo delle routine Sub, è importante sapere come

eseguirle. Una routine Sub è inutile, se non si sa come eseguirla.

A proposito, si dice *eseguire* una routine Sub ma anche *lanciare* o *richiamare*. Potete usare il termine che preferite.

Potete *eseguire* una routine Sub in molti modi; questa è una delle ragioni per cui potete fare tante cose utili con le routine Sub. Ecco un elenco completo dei modi in cui potete eseguire una routine Sub:

- In VBE scegliete Esegui ➪ Esegui Sub/UserForm. Excel eseguirà la routine Sub in cui si trova il cursore. Questo comando ha anche altre due alternative: il tasto F5 e il pulsante Esegui Sub/UserForm sulla barra degli strumenti Standard di VBE. Questi metodi non funzionano se la routine richiede uno o più argomenti.
- Usate la finestra di dialogo Macro di Excel. Aprite questa finestra selezionando il comando Sviluppo ➪ Codice ➪ Macro o scegliendo Visualizza ➪ Macro ➪ Macro. Oppure ignorate del tutto la barra multifunzione e premete Alt + F8. Quando verrà visualizzata la finestra di dialogo Macro, selezionate la routine Sub desiderata e fate clic su Esegui. Questa finestra di dialogo elenca solo le routine che non richiedono un argomento.
- Premete la combinazione Ctrl + *tasto* (o Ctrl + Maiusc + *tasto*) che avete assegnato alla routine Sub (sempre se gliene avete assegnato uno).
- Fate clic su un pulsante o una forma nel foglio di lavoro. Al pulsante o alla forma deve essere assegnata una routine Sub, cosa molto semplice da fare.
- Da un'altra routine Sub.
- Fate clic su un pulsante che avete aggiunto alla barra degli strumenti Accesso rapido (vedi Capitolo 19).
- Da un oggetto personalizzato che avete aggiunto alla barra multifunzione (vedi Capitolo 19.)
- Quando si verifica un evento, come spiegato più avanti nel Capitolo 11, come l'apertura, la chiusura, il salvataggio della cartella di lavoro, la modifica di una cella, l'attivazione di un foglio e altro.
- Dalla finestra Immediata di VBE. Basta digitare il nome della routine Sub e premere Invio.

Alcune di queste tecniche sono trattate nei paragrafi che seguono. Per seguirle, è necessario inserire una routine Sub in un modulo VBA:

1. **Create una nuova cartella di lavoro.**
2. **Premete Alt + F11 per attivare VBE.**
3. **Selezionate la cartella di lavoro nel riquadro Progetto.**
4. **Scegliete Inserisci ➪ Modulo per inserire un nuovo modulo.**
5. **Inserite quanto segue nel modulo:**

```
Sub ShowCubeRoot()
      Num = InputBox("Inserisci un numero positivo")
      MsgBox Num ^ (1/3) & " è la sua radice cubica."
End Sub
```

Questa routine richiede all'utente un numero e poi visualizza la radice cubica di quel numero in una finestra. Le Figure 5.1 e 5.2 mostrano quello che succede quando si esegue questa routine.

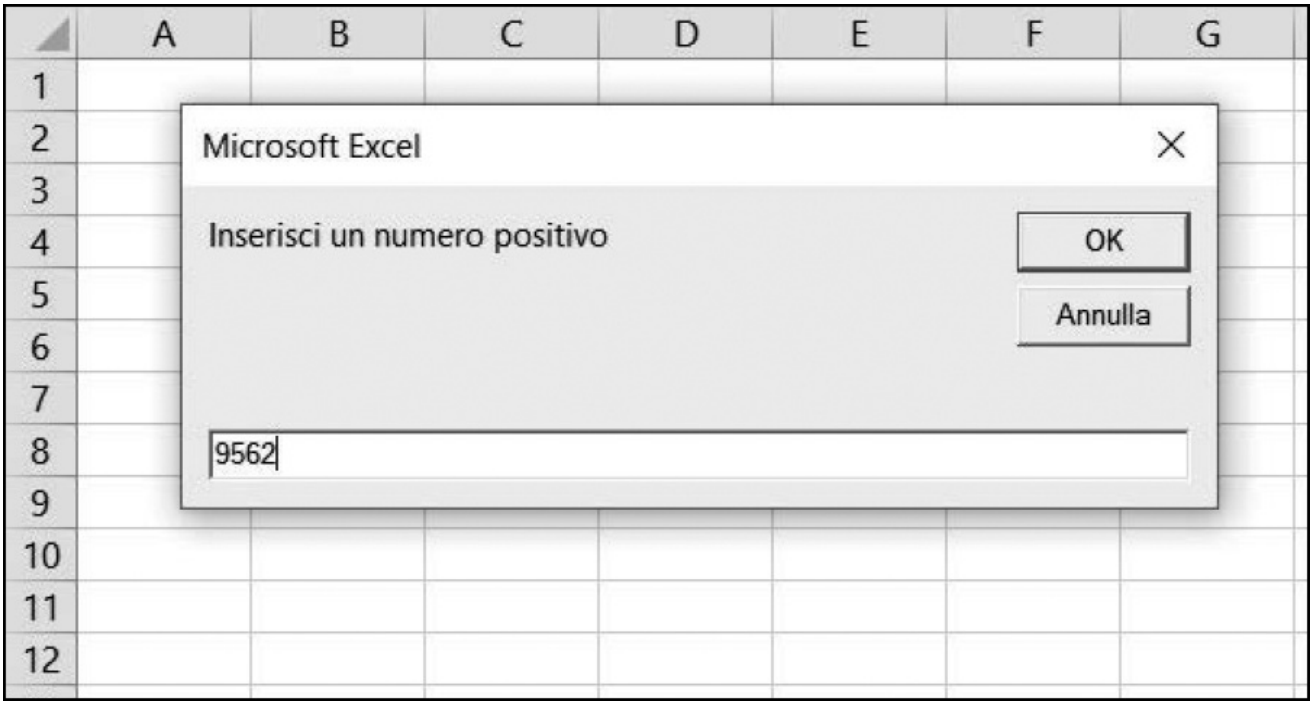

FIGURA 5.1 Uso della funzione VBA InputBox per ottenere un numero.

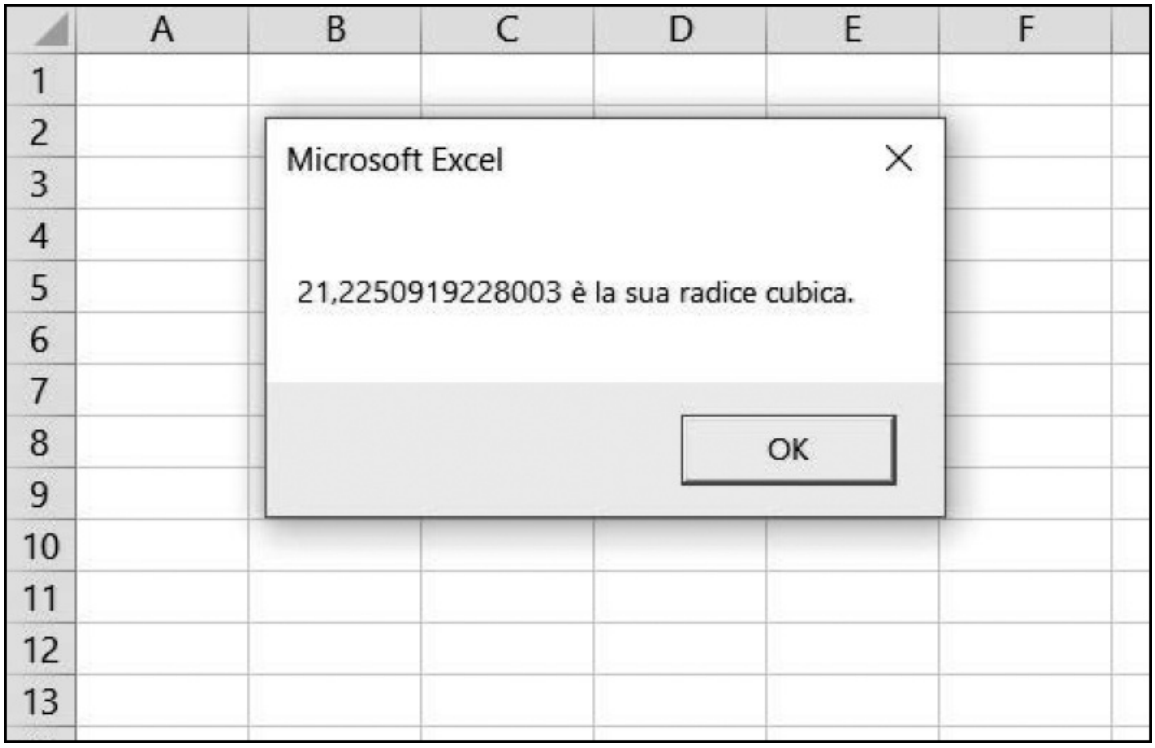

FIGURA 5.2 Visualizzazione della radice cubica di un numero tramite la funzione MsgBox.

Comunque, ShowCubeRoot non è un buon esempio di macro. Non controlla gli errori, quindi è facile che si blocchi. Provate a fare clic sul pulsante Annulla nella casella di immissione o a inserire un numero negativo. Entrambe le azioni genereranno un messaggio d'errore. Il Capitolo 12 spiega come gestire questi tipi di errori.

Eseguire direttamente una routine Sub

Un modo per eseguire questa routine è farlo direttamente dal modulo VBA in cui è stata definita. Seguite questi passi.

1. **Attivate il VBE e selezionate il modulo VBA che contiene la routine.**
2. **Posizionate il cursore dentro il codice della routine.**
3. **Premete F5 (o selezionate il comando Esegui ➪ Esegui Sub/UserForm).**
4. **Rispondete alla casella di input e fate clic su OK.**

   La routine mostrerà la radice cubica del numero inserito.

RICORDA!

Non potete utilizzare Esegui ➪ Esegui Sub/UserForm per eseguire una routine Sub che utilizza argomenti, poiché non avrete modo di passare argomenti alla routine. Se la routine richiede uno o più argomenti, l'unico modo per eseguirla è richiamarla da un'altra routine, che deve fornire l'argomento o gli argomenti da essa richiesti.

Eseguire una routine dalla finestra di dialogo Macro

Nella maggior parte dei casi, le routine Sub si eseguono da Excel, non dal VBE. I prossimi passaggi descrivono come eseguire una macro utilizzando la finestra di dialogo Macro di Excel.

1. **In VBE, attivate Excel.**

   La via più veloce è premere Alt + F11.
2. **Selezionate il comando Sviluppo ➪ Codice ➪ Macro (o premete Alt + F8).**

   Excel visualizzerà la finestra di dialogo rappresentata nella Figura 5.3.
3. **Selezionate la macro.**

   Fate clic su **Esegui** (o fate doppio clic sul nome della macro nella casella di riepilogo).

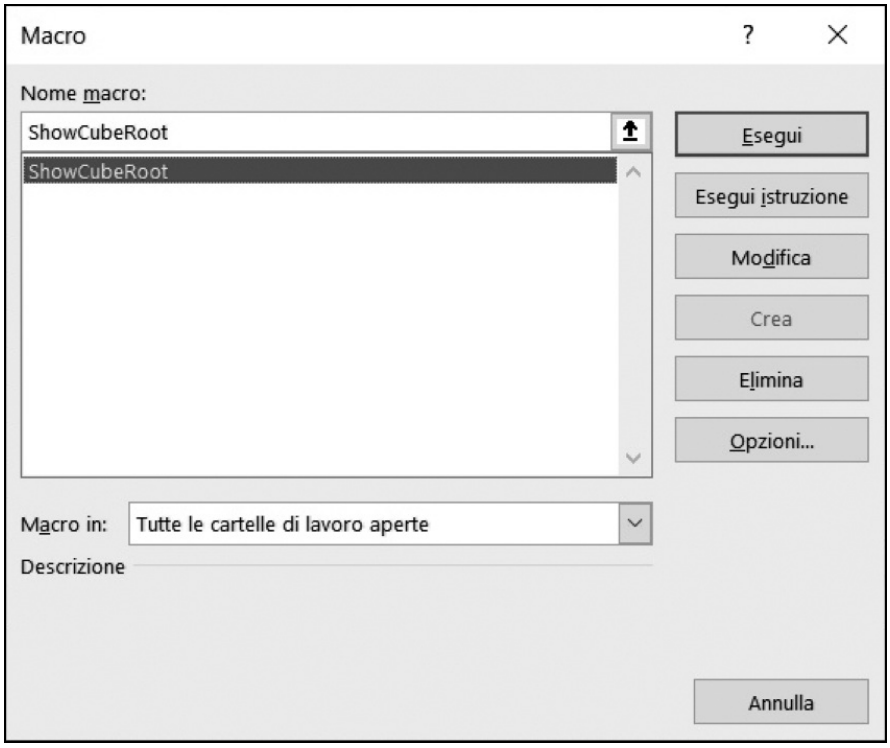

FIGURA 5.3 La finestra di dialogo Macro elenca tutte le routine Sub disponibili.

RICORDA!

La finestra di dialogo Macro non visualizza le routine Sub che usano argomenti. Questo perché qui non potete specificare gli argomenti.

Eseguire una macro utilizzando un tasto di scelta rapida

Un altro modo per eseguire una macro consiste nell'usare un tasto di scelta rapida. Ma prima di poter utilizzare questo metodo, dovrete assegnare un tasto di scelta rapida alla macro.

Avete la possibilità di assegnare un tasto di scelta rapida nella finestra di dialogo Registra macro quando registrate la macro. Se create la routine senza utilizzare il registratore di macro, potrete assegnare un tasto di scelta rapida (o cambiare il tasto di scelta rapida esistente) utilizzando la seguente routine.

1. **Selezionate il comando Sviluppo ➪ Codice ➪ Macro.**
2. **Selezionate il nome della routine Sub dalla casella di riepilogo.**

   In questo esempio, la routine è denominata ShowCubeRoot.
3. **Fate clic sul pulsante Opzioni.**

   Excel visualizzerà la finestra di dialogo Opzioni macro mostrata in Figura 5.4.
4. **Fate clic sull'opzione Tasto di scelta rapida e inserite una lettera nella casella con l'indicazione Ctrl +.**

   La lettera inserita corrisponde alla combinazione di tasti che desiderate utilizzare per l'esecuzione della macro. Per esempio, se inserite la lettera c minuscola, potrete eseguire la macro premendo Ctrl + C. Se inserite una lettera maiuscola, dovrete aggiungere il tasto Maiusc alla combinazione di tasti. Per esempio, se inserite C, potrete eseguire la macro premendo Ctrl + Maiusc + C.
5. **Fate clic su OK per chiudere la finestra di dialogo Opzioni macro e poi fate clic su Annulla per chiudere la finestra di dialogo Macro.**

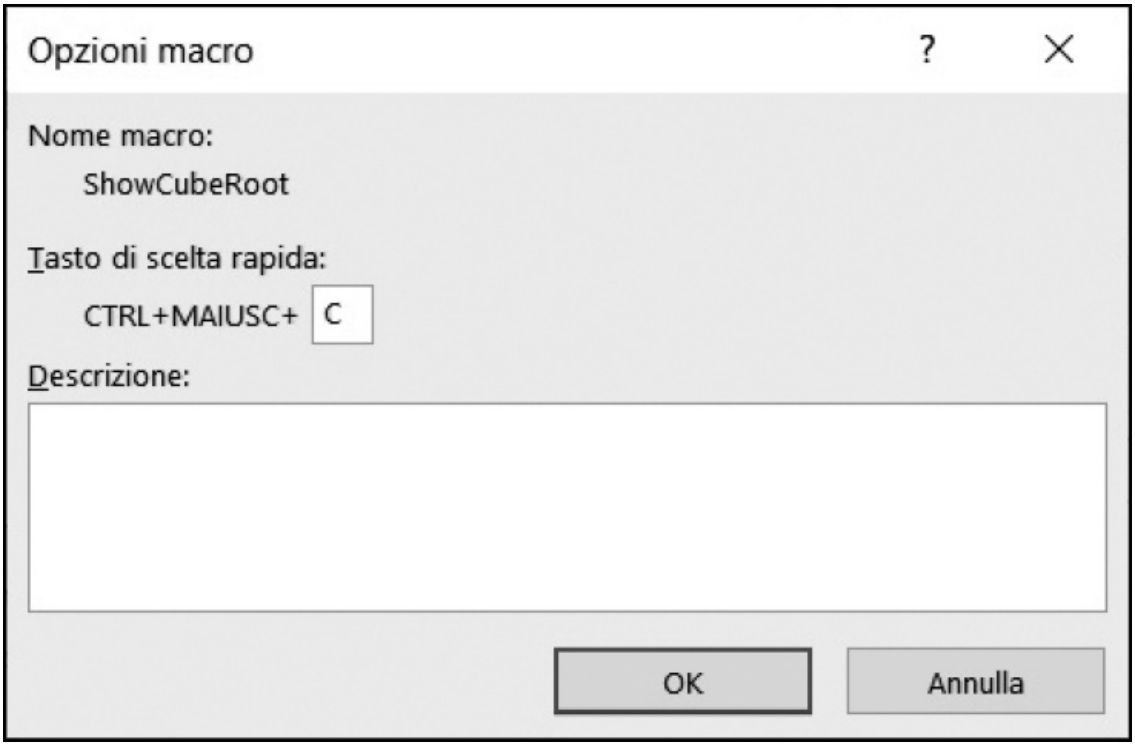

FIGURA 5.4 La finestra di dialogo Opzioni macro consente di impostare le opzioni per le vostre macro.

Dopo aver assegnato un tasto di scelta rapida, potrete premere quella combinazione di tasti per eseguire la macro. Un tasto di scelta rapida non funziona se è assegnato a una macro che utilizza un argomento.

WARNING

I tasti di scelta rapida assegnati alle macro sostituiscono i tasti di scelta rapida standard di Excel. Per esempio, Ctrl + C è il tasto di scelta rapida standard per copiare i dati. Se assegnate Ctrl + C a una macro, non potrete più usare Ctrl + C per il comando Copia. Questo di solito non è un grosso problema, perché Excel offre quasi sempre più modi per eseguire i comandi.

Eseguire la routine da un pulsante o da una forma

A volte, vi potrebbe piacere l'idea di assegnare la macro a un pulsante (o a qualsiasi altra forma) su un foglio di lavoro. Per assegnare la macro a un pulsante, seguite questa procedura.

1. **Attivate un foglio di lavoro.**
2. **Aggiungete un pulsante dal gruppo Controlli modulo.**

   Per visualizzare il gruppo Controlli modulo, selezionate il comando Sviluppo ➪ Controlli ➪ Inserisci (vedi Figura 5.5).

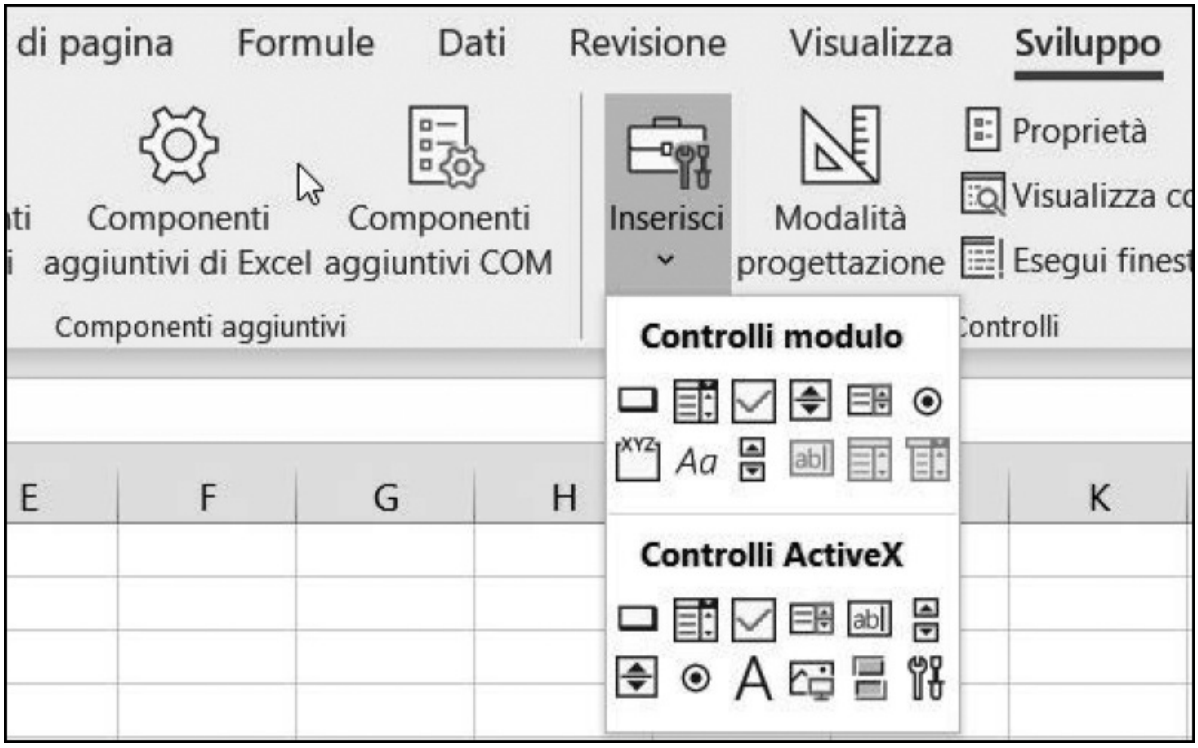

FIGURA 5.5 La barra multifunzione mostra i controlli disponibili.

3. **Fate clic sullo strumento Pulsante nel gruppo Controlli modulo.**

   È la prima opzione nella prima riga di controlli.

4. **Trascinatelo nel foglio di lavoro, per creare il pulsante.**

   Dopo aver aggiunto il pulsante al foglio di lavoro, Excel vi legge nel pensiero e visualizza la finestra di dialogo Assegna macro, rappresentata nella Figura 5.6.

5. **Selezionate la macro che volete assegnare al pulsante.**
6. **Fate clic su OK.**

Dopo aver eseguito l'assegnazione, fate clic sul pulsante ed eseguirete la macro, proprio come per magia.

RICORDA!

Quando aggiungete un pulsante, tenete presente che la casella a discesa mostra due gruppi di controlli: Controlli modulo e Controlli ActiveX. Questi due gruppi di controlli sono simili, ma in realtà sono molto diversi. In pratica, i Controlli modulo sono più facili da usare.

FIGURA 5.6 Quando aggiungete un pulsante a un foglio di lavoro, Excel visualizza automaticamente la finestra di dialogo Assegna macro.

Potete anche assegnare una macro a una qualsiasi altra forma o oggetto. Per esempio, supponiamo che desideriate eseguire una macro quando l'utente fa clic su un oggetto Rettangolo. Seguite questa procedura.

1. **Aggiungete il rettangolo al foglio di lavoro.**

   Inserite un rettangolo scegliendo Inserisci ➪ Illustrazioni ➪ Forme.

2. **Fate clic destro sul rettangolo.**
3. **Scegliete Assegna macro dal menu contestuale.**
4. **Selezionate la macro nella finestra di dialogo Assegna macro.**
5. **Fate clic su OK.**

Dopo aver eseguito questi passaggi, facendo clic sul rettangolo verrà eseguita la macro che gli avete assegnato.

Eseguire la routine da un'altra routine

Potete anche eseguire una routine da un'altra routine, nel seguente modo.

1. **Attivate il modulo VBA che contiene la routine ShowCubeRoot.**
2. **Inserite questa nuova routine (sopra o sotto il codice di ShowCubeRoot, non fa differenza):**

```
Sub NewSub()
     Call ShowCubeRoot
End Sub
```

3. **Eseguite la macro NewSub.**

   Il modo più semplice per farlo è posizionare il cursore ovunque all'interno del codice di NewSub e premere F5. Notate che questa routine NewSub non fa altro che eseguire la routine ShowCubeRoot.

A questo proposito, la parola chiave Call è opzionale. La dichiarazione può essere costituita anche solo dal nome della routine Sub. Tuttavia, l'uso della parola chiave Call rende più chiaro il fatto che viene richiamata una routine.

Eseguire le routine Function

Le funzioni, a differenza delle routine Sub, possono essere eseguite in due soli modi:

- richiamando la funzione da un'altra routine Sub o Function;
- usando la funzione in una formula del foglio di lavoro.

Provate questa semplice funzione. Inseritela in un modulo VBA:

```
Function CubeRoot(number)
     CubeRoot = number ^ (1 / 3)
End Function
```

Questa funzione è piuttosto scarna; non fa altro che calcolare la radice cubica del numero passatole come argomento. Tuttavia, rappresenta un punto di partenza per la comprensione delle funzioni. Illustra anche un concetto importante delle funzioni: come restituire un valore (ve lo ricordate che una funzione restituisce un valore, vero?).

Notate che la singola riga di codice che costituisce questa routine Function esegue un calcolo. Il risultato del calcolo matematico (number elevato alla potenza 1/3) viene assegnato alla variabile CubeRoot. Non a caso, CubeRoot è anche il nome della funzione. Per dire alla funzione quale valore deve restituire, basta assegnare tale valore al nome della funzione.

Richiamare la funzione da una routine Sub

Poiché non potete eseguire direttamente una funzione, è necessario richiamarla da un'altra routine. Immettere la seguente semplice routine nello stesso modulo VBA che contiene la funzione CubeRoot:

```
Sub CallerSub()
     Ans = CubeRoot(125)
     MsgBox Ans
End Sub
```

Quando eseguite la routine CallerSub (utilizzando uno dei metodi descritti in precedenza in questo capitolo), Excel visualizza una finestra che contiene il valore della variabile Ans, che è 5.

Ecco che cosa sta succedendo: la funzione CubeRoot viene eseguita e riceve l'argomento 125. Il calcolo viene eseguito dal codice della funzione (utilizzando il valore passato come argomento) e il valore restituito dalla funzione viene poi assegnato alla variabile Ans. La funzione MsgBox, infine, visualizza il valore della variabile Ans.

Provate a cambiare l'argomento passato alla funzione CubeRoot ed eseguite di nuovo la macro CallerSub. Funziona esattamente come dovrebbe, sempre che alla funzione venga passato un argomento valido (un numero positivo).

La routine CallerSub potrebbe essere semplificata un po'. La variabile Ans non è davvero necessaria, a meno che il codice non utilizzi quella variabile in seguito. Per ottenere lo stesso risultato potete anche utilizzare questa singola istruzione:

```
MsgBox CubeRoot(125)
```

Richiamare una funzione da una formula del foglio di lavoro

Ora è il momento di chiamare questa routine Function da una formula del foglio di lavoro. Attivate un foglio di lavoro nella stessa cartella di lavoro che contiene la definizione della funzione CubeRoot. Quindi inserite, in qualsiasi cella, la seguente formula:

```
=CubeRoot(1728)
```

La cella visualizza 12, che è, in effetti, la radice cubica di 1728.

Come potreste aspettarvi, potete anche utilizzare un riferimento a una cella come argomento per la funzione CubeRoot. Se la cella A1 contiene un valore, per esempio, potete immettere **=CubeRoot(A1)**.

In questo caso, la funzione restituisce il numero ottenuto calcolando la radice cubica del valore contenuto nella cella A1.

In un foglio di lavoro, potete utilizzare questa funzione un numero illimitato di volte. Come le funzioni integrate in Excel, anche le funzioni personalizzate vengono visualizzate nella finestra di dialogo Inserisci funzione. Fate clic sul pulsante Inserisci funzione della barra degli strumenti e selezionate la categoria Definite dall'utente. Come potete vedere nella Figura 5.7, la finestra di dialogo Inserisci funzione elenca anche la vostra nuova funzione.

FIGURA 5.7 La funzione CubeRoot viene visualizzata nella categoria Definite dall'utente della finestra di dialogo Inserisci funzione.

CONSIGLIO

Se desiderate che la finestra di dialogo Inserisci funzione visualizzi una descrizione della funzione, ecco come potete fare.

1. **Selezionate il comando Sviluppo ➪ Codice ➪ Macro.**

   Excel visualizza la finestra di dialogo Macro, ma CubeRoot non viene visualizzata nell'elenco (CubeRoot è una routine Function e questo elenco mostra solo le routine Sub). Ma non preoccupatevi.

2. **Nella casella Nome macro digitate CubeRoot.**
3. **Fate clic sul pulsante Opzioni.**
4. **Inserite una descrizione della funzione nella casella Descrizione.**
5. **Fate clic su OK per chiudere la finestra di dialogo Opzioni macro.**
6. **Chiudete la finestra di dialogo Macro facendo clic sul pulsante Annulla.**

   Questo testo descrittivo ora verrà visualizzato nella finestra di dialogo Inserisci funzione.

La Figura 5.8 mostra la funzione CubeRoot utilizzata nelle formule del foglio di lavoro.

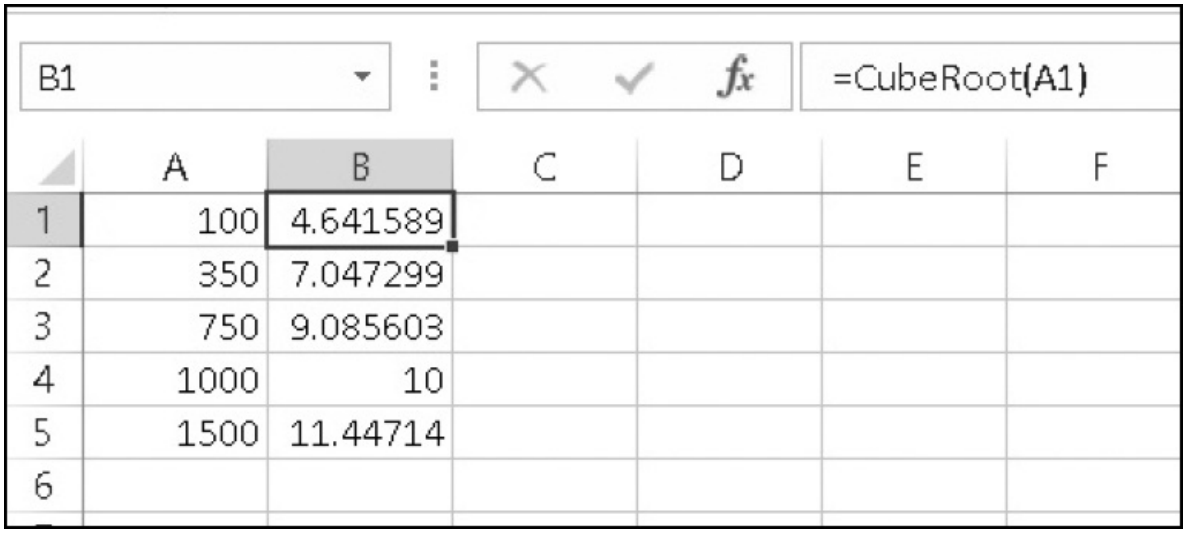

FIGURA 5.8 Utilizzo della funzione CubeRoot nelle formule.

Adesso le cose potrebbero iniziare a venire da sè. Avete trovato molte informazioni sulle routine Sub e Function. Nel Capitolo 6, inizieremo a creare delle macro oltre a illustrare i dettagli sullo sviluppo di macro utilizzando il registratore di macro di Excel. Il Capitolo 20 svelerà ancora più informazioni sulle routine Function.

Capitolo 6

Uso del registratore di macro di Excel

IN QUESTO CAPITOLO

- **Registrare le azioni utilizzando il registratore di macro incorporato**
- **I tipi di macro che potete registrare**
- **Impostare le opzioni appropriate per la registrazione macro**
- **Valutare l'efficienza della macro registrata**

Potete impiegare due metodi per creare una macro di Excel:

- registrarla utilizzando il registratore di macro di Excel;
- scriverla manualmente.

In questo capitolo vedremo in dettaglio l'uso del registratore di macro di Excel. Registrare una macro non è sempre l'approccio migliore, e alcune macro, semplicemente, non possono essere registrate, indipendentemente da quanto ci proviate e riproviate. Tuttavia, come vedrete, il registratore di macro di Excel è molto utile. Anche se la macro registrata non fa esattamente ciò che desiderate, il registratore di macro può quasi sempre guidarvi nella giusta direzione.

Concetti di base sulla registrazione

Per registrare una macro seguite i seguenti passi base.

1. **Determinate in anticipo quello che volete fare con la macro.**
2. **Preparate le cose correttamente.**

   Questo passaggio determina il buon funzionamento della vostra macro.
3. **Determinate se desiderate che i riferimenti di cella nella vostra macro siano relativi o assoluti.**
4. **Fate clic sul pulsante Registra macro sul lato sinistro della barra di stato (o selezionate il comando Sviluppo ➪ Codice ➪ Registra macro).**

   Excel visualizzerà la finestra di dialogo Registra macro.
5. **Immettete un nome, un tasto di scelta rapida, una posizione per la macro e una descrizione.**

   Ognuno di questi elementi, a eccezione del nome, è facoltativo.
6. **Nella finestra di dialogo Registra macro fate clic su OK.**

   Excel inserisce automaticamente un modulo VBA nella cartella di lavoro specificata nella casella Memorizza macro in. Da questo momento in poi, Excel convertirà le vostre azioni in codice VBA.
   Nella barra di stato troverete anche il pulsante Interrompi registrazione.
7. **Eseguite le azioni che desiderate registrare, usando il mouse o la tastiera.**
8. **Al termine, fate clic sul pulsante Interrompi registrazione, sulla barra di stato (o selezionate il comando Sviluppo ➪ Codice ➪ Interrompi registrazione).**

   Excel smetterà di registrare le vostre azioni.
9. **Provate la macro per assicurarvi che funzioni correttamente.**

   (Facoltativo) Pulite il codice rimuovendo le dichiarazioni estranee o aggiungetegli alcuni commenti

10. per spiegarne il funzionamento.

Il registratore di macro è più adatto per macro semplici e intuitive. Per esempio, potreste aver bisogno di una macro che applichi la formattazione a un intervallo di celle selezionato o che configuri le intestazioni di righe e colonne di un nuovo foglio di lavoro.

RICORDA!

Il registratore di macro è adatto solo a realizzare routine Sub. Non potete utilizzarlo per creare routine Function.

Inoltre potreste trovare il registratore di macro utile per lo sviluppo di macro più complesse. Vale a dire: potete registrare alcune azioni e poi copiare il codice registrato in un'altra macro, più complessa. Nella maggior parte dei casi, infatti, è necessario modificare il codice registrato e aggiungergli alcune nuove istruzioni VBA.

Il registratore di macro non può generare codice per nessuna delle seguenti attività, descritte più avanti nel libro.

- Esecuzione di qualsiasi tipo di ciclo ripetitivo.
- Esecuzione di qualsiasi tipo di azione condizionale (che utilizza un'istruzione If-Then).
- Assegnazione di valori a variabili.
- Indicazione del tipo dei dati.
- Visualizzazione di messaggi pop-up.
- Visualizzazione di finestre di dialogo personalizzate.

RICORDA!

La limitata capacità del registratore di macro non riduce di certo la sua importanza. Registrare azioni è forse il modo migliore per padroneggiare VBA. In caso di dubbi, provate a registrare i passaggi. Anche se il risultato potrebbe non essere esattamente quello che desiderate, la visualizzazione del codice registrato potrebbe rivelarvi alcuni oggetti, proprietà e metodi di cui non eravate a conoscenza e indirizzarvi nella giusta direzione.

Prepararsi alla registrazione

Prima di compiere il grande passo e accendere il registratore di macro, dedicate qualche minuto a pensare a che cosa fare. Dovete registrare una macro in modo che Excel possa ripetere automaticamente le azioni registrate, quindi volete che quelle azioni siano accurate.

RICORDA!

In definitiva, il successo di una macro registrata dipende da cinque fattori.

- Come è impostata la cartella di lavoro mentre registrate la macro.
- Che cosa è selezionato quando iniziate la registrazione.
- Se utilizzate la modalità di registrazione assoluta o relativa.
- La precisione delle azioni registrate.
- Il contesto in cui riprodurrete la macro registrata.

L'importanza di questi fattori sarà più chiara mano a mano che procederete nel processo di registrazione.

Relativo o assoluto?

Quando registra le vostre azioni, Excel normalmente registra i riferimenti assoluti alle celle (questa è la modalità di registrazione predefinita). Ma molto spesso, questa modalità di registrazione è errata. Se utilizzate la modalità di registrazione assoluta, Excel registra i riferimenti esattamente a quelle celle e solo a quelle.

Con la registrazione relativa, Excel registra i riferimenti per una posizione qualsiasi, relativamente alle celle usate. Vi sfugge la differenza? Continuate a leggere.

Registrare in modalità assoluta

Aprite una nuova cartella di lavoro e seguite questi passaggi per registrare una semplice macro in modalità assoluta. Questa macro non fa altro che inserire in un foglio di lavoro i nomi di tre mesi.

1. **Assicuratevi che il pulsante Sviluppo ➪ Codice ➪ Usa riferimenti relativi non sia evidenziato e poi selezionate il comando Sviluppo ➪ Codice ➪ Registra macro.**
2. **Come nome per questa macro, digitate Absolute.**
3. **Fate clic su OK per avviare la registrazione.**
4. **Attivate la cella B1 e digitatevi Gen.**
5. **Spostatevi sulla cella C1 e digitate Feb.**
6. **Spostatevi sulla cella D1 e digitate Mar.**
7. **Fate clic sulla cella B1 per attivarla, di nuovo.**
8. **Arrestate il registratore di macro.**
9. **Premete Alt + F11 per attivare il VBE.**
10. **Esaminate il modulo Modulo1.**

    Excel avrà generato il seguente codice:

```
Sub Absolute()
'
' Absolute Macro
'
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Gen"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "Feb"
    Range("D1").Select
    ActiveCell.FormulaR1C1 = "Mar"
    Range("B1").Select
End Sub
```

    Quando questa macro viene eseguita, seleziona la cella `B1` e inserisce i nomi dei tre mesi nell'intervallo `B1:D1`. Quindi la macro riattiva la cella `B1`.

Queste stesse azioni si verificano indipendentemente dalla cella attiva nel momento in cui eseguite la macro. Una macro registrata utilizzando riferimenti assoluti, quando viene eseguita produce sempre gli stessi risultati. In questo caso, la macro immette sempre i nomi dei primi tre mesi e sempre nell'intervallo `B1:D1` nel foglio di lavoro attivo.

Registrare in modalità relativa

In alcuni casi, desiderate che la macro registrata funzioni con posizioni delle celle relative. A volte volete che la macro inizi a immettere i nomi dei mesi nella cella attualmente attiva. In tal caso, dovrete utilizzare la registrazione relativa.

Potete modificare il modo in cui Excel registra le azioni facendo clic sul pulsante Usa riferimenti relativi, nel gruppo Codice della scheda Sviluppo. Si tratta di un pulsante a commutatore: on/off. Quando il pulsante appare evidenziato in un colore diverso, la modalità di registrazione è relativa. Quando il pulsante è "normale", state registrando in modalità assoluta.

Potete modificare il metodo di registrazione in qualsiasi momento, anche durante la registrazione.

Per vedere come funziona la registrazione in modalità relativa, eliminate il contenuto dell'intervallo di celle `B1:D1` ed eseguite le seguenti operazioni.

1. **Attivate la cella B1.**
2. **Selezionate il comando Sviluppo ➪ Codice ➪ Registra macro.**
3. **Date un nome a questa macro.**
4. **Fate clic su OK per avviare la registrazione.**
5. **Fate clic sul pulsante Usa riferimenti relativi per cambiare la modalità di registrazione e attivare quella relativa.**

    Quando fate clic su questo pulsante, viene visualizzato in un colore diverso dal resto della barra multifunzione.
6. **Nella cella B1 digitate Gen.**
7. **Spostatevi nella cella C1 e digitate Feb.**
8. **Spostatevi nella cella D1 e digitate Mar.**

9. **Selezionate la cella B1.**
10. **Arrestate il registratore di macro.**

Notate che questa routine differisce leggermente dall'esempio precedente. In questo esempio, prima di iniziare la registrazione viene attivata la cella iniziale. Questo è un passaggio importante quando registrate macro che utilizzano come base la cella attiva.

Questa macro inizia sempre a inserire il testo nella cella attiva. Provate. Portate il cursore della cella su qualsiasi cella e poi eseguite la macro relativa. I nomi dei mesi vengono sempre inseriti a partire dalla cella attiva.

Con la modalità di registrazione relativa, il codice generato da Excel è molto diverso da quello generato in modalità assoluta:

```
Sub Relative()
'
' Relative Macro
'
     ActiveCell.FormulaR1C1 = "Gen"
     ActiveCell.Offset(0, 1).Range("A1").Select
     ActiveCell.FormulaR1C1 = "Feb"
     ActiveCell.Offset(0, 1).Range("A1").Select
     ActiveCell.FormulaR1C1 = "Mar"
     ActiveCell.Offset(0, -2).Range("A1").Select
End Sub
```

Per provare questa macro, rendete attiva una qualsiasi cella, diversa da `B1`. I nomi dei mesi vengono inseriti nelle tre celle, a partire dalla cella che avete attivato.

PER I PIÙ CURIOSI

Notate che il codice generato dal registratore di macro fa riferimento alla cella `A1`. Questo può sembrare strano, perché durante la registrazione della macro non avete mai usato la cella `A1`. Questo è semplicemente un sottoprodotto del modo in cui funziona il registratore di macro (questo argomento verrà discusso in modo più dettagliato nel Capitolo 8, dove verrà esplorata la proprietà Offset).

Che cosa viene registrato?

Quando avviate il registratore di macro, Excel converte le azioni del mouse e della tastiera in codice VBA. Il modo migliore per comprendere il processo consiste nel guardare il registratore di macro in azione (vedi Figura 6.1.)

FIGURA 6.1 Una disposizione delle finestre comoda per studiare il funzionamento del registratore di macro.

Seguite questi passaggi.

1. **Iniziate con una cartella di lavoro vuota.**
2. **Assicuratevi che la finestra di Excel non sia ingrandita.**
3. **Premete Alt + F11 per attivare il VBE (e assicuratevi che *questa* finestra del programma non sia ingrandita).**
4. **Ridimensionate e disponete la finestra di Excel e la finestra di VBE in modo che siano entrambe visibili.**

   Per risultati ottimali, posizionate la finestra di Excel sopra la finestra di VBE e riducete al minimo tutte le altre applicazioni in esecuzione.
5. **Attivate Excel e selezionate il comando Sviluppo ➪ Codice ➪ Registra macro.**
6. **Fate clic su OK per avviare il registratore di macro.**

   Excel inserisce un nuovo modulo (denominato Modulo1) e inizia a registrare in quel modulo.
7. **Attivate la finestra del programma VBE.**
8. **Nel riquadro Progetto, fate doppio clic su Modulo1 per visualizzare quel modulo nella finestra del codice.**

Tornate in Excel e "giocateci" un po'. Scegliete vari comandi Excel e osservate il codice generato nella finestra di VBE. Selezionate celle, inserite dati, formattate le celle, usate i comandi della barra multifunzione, create un grafico, modificate la larghezza delle colonne, manipolate gli oggetti grafici e così via: quello che volete! Sarà vantastico vedere Excel mentre genera il codice VBA davanti ai vostri occhi.

CONSIGLIO

Se disponete di due monitor, potrebbe essere utile mantenere Excel su un monitor e la finestra di VBE sull'altro.

Opzioni di registrazione

Quando registrate le vostre azioni per creare il codice VBA, avete a disposizione diverse opzioni. Ricordate che Sviluppo ➪ Codice ➪ Registra macro mostra la finestra di dialogo Registra macro prima che inizi la registrazione, come illustrato nella Figura 6.2.

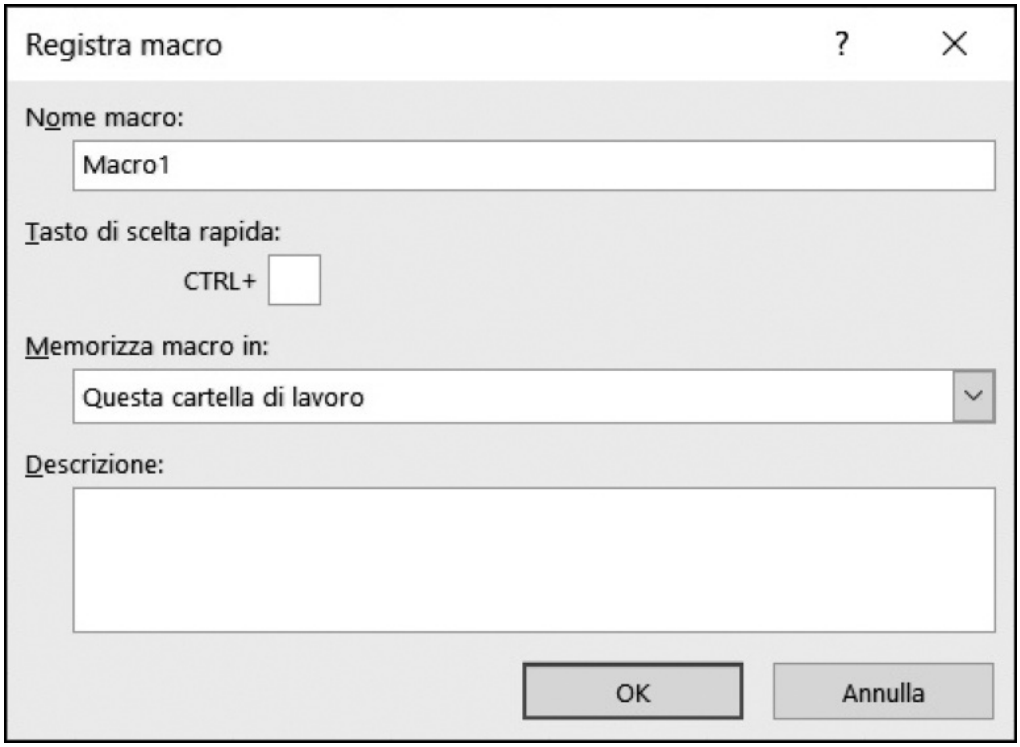

FIGURA 6.2 La finestra di dialogo Registra macro fornisce diverse opzioni.

La finestra di dialogo Registra macro vi consente di specificare alcuni aspetti della macro. I prossimi paragrafi vi guidano in queste opzioni.

Nome macro

Potete inserire un nome per la routine Sub che state registrando. Come impostazione predefinita, per ogni macro che registrate, Excel utilizza i nomi Macro1, Macro2 e così via. Non preoccupatevi se non avete assegnato un nome descrittivo alla macro nella finestra di dialogo Registra macro. Potete sempre darle un nome più descrittivo in seguito, modificando il codice registrato in VBE.

Tasto di scelta rapida

L'opzione Tasto di scelta rapida vi consentirà di eseguire la macro premendo una combinazione di tasti. Per esempio, se digitate w (in minuscolo), potrete eseguire la macro premendo Ctrl + W. Se digitate W (in maiuscolo), la macro si attiva quando premete Ctrl + Maiusc + W.

Potete aggiungere o modificare un tasto di scelta rapida in qualsiasi momento, quindi non c'è un vero motivo per impostare questa opzione quando registrate la macro. Per istruzioni sull'assegnazione di un tasto di scelta rapida a una macro esistente, consultate il Capitolo 5.

L'opzione Memorizza macro in

L'opzione Memorizza macro in indica a Excel dove memorizzare la macro che state registrando. Per impostazione predefinita, Excel inserisce la macro registrata in un modulo della cartella di lavoro attiva. Se preferite, potete registrarla in una nuova cartella di lavoro (Excel aprirà una nuova cartella di lavoro vuota) o nella cartella di lavoro Cartella macro personale.

La Cartella macro personale è una cartella di lavoro nascosta che si apre automaticamente all'avvio di Excel. È un buon posto per memorizzare le macro che pensate di usare con più cartelle di lavoro. La Cartella macro personale è denominata PERSONAL.XLSB. Questo file non esiste finché non lo si specifica come destinazione di una macro registrata. Se avete apportato modifiche a questo file, Excel vi chiederà di salvarlo, all'uscita.

Descrizione

Se desiderate aggiungere alcuni commenti descrittivi alla macro, inseriteli nella casella Descrizione. Potete scrivere tutto quello che volete, o anche niente del tutto.

Questa cosa è efficiente?

Potreste pensare che la registrazione di una macro generi codice VBA eccellente, migliore di quello che potreste mai scrivere manualmente. No, non è così. Poiché il registratore di macro deve essere abbastanza generico da registrare virtualmente ogni combinazione di azioni, spesso genera codice di troppo: funziona, ma non è molto efficiente.

Per dimostrare quanto sia inefficiente il codice generato dal registratore di macro, fate la seguente prova.

1. **Avviate il registratore di macro.**
2. **Selezionate il comando Layout di pagina ➪ Imposta pagina ➪ Orientamento ➪ Orizzontale.**
3. **Fermate il registratore di macro.**

Per dare un'occhiata alla macro, attivate il foglio Modulo1. Questo singolo (e molto semplice) comando ha generato il seguente codice:

```
Sub Macro1()
Application.PrintCommunication = False
     With ActiveSheet.PageSetup
          .PrintTitleRows = ""
          .PrintTitleColumns = ""
     End With
     Application.PrintCommunication = True
     ActiveSheet.PageSetup.PrintArea = ""
     Application.PrintCommunication = False
     With ActiveSheet.PageSetup
          .LeftHeader = ""
          .CenterHeader = ""
          .RightHeader = ""
          .LeftFooter = ""
          .CenterFooter = ""
          .RightFooter = ""
          .LeftMargin = Application.InchesToPoints(0.7)
          .RightMargin = Application.InchesToPoints(0.7)
          .TopMargin = Application.InchesToPoints(0.75)
          .BottomMargin = Application.InchesToPoints(0.75)
          .HeaderMargin = Application.InchesToPoints(0.3)
          .FooterMargin = Application.InchesToPoints(0.3)
          .PrintHeadings = False
          .PrintGridlines = False
          .PrintComments = xlPrintNoComments
```

```
        .PrintQuality = 600
        .CenterHorizontally = False
        .CenterVertically = False
        .Orientation = xlLandscape
        .Draft = False
        .PaperSize = xlPaperLetter
        .FirstPageNumber = xlAutomatic
        .Order = xlDownThenOver
        .BlackAndWhite = False
        .Zoom = 100
        .PrintErrors = xlPrintErrorsDisplayed
        .OddAndEvenPagesHeaderFooter = False
        .DifferentFirstPageHeaderFooter = False
        .ScaleWithDocHeaderFooter = True
        .AlignMarginsHeaderFooter = True
        .EvenPage.LeftHeader.Text = ""
        .EvenPage.CenterHeader.Text = ""
        .EvenPage.RightHeader.Text = ""
        .EvenPage.LeftFooter.Text = ""
        .EvenPage.CenterFooter.Text = ""
        .EvenPage.RightFooter.Text = ""
        .FirstPage.LeftHeader.Text = ""
        .FirstPage.CenterHeader.Text = ""
        .FirstPage.RightHeader.Text = ""
        .FirstPage.LeftFooter.Text = ""
        .FirstPage.CenterFooter.Text = ""
        .FirstPage.RightFooter.Text = ""
    End With
    Application.PrintCommunication = True
End Sub
```

Sarete sicuramente sorpresi dalla quantità di codice generato da questo singolo comando. Sebbene abbiate modificato una sola impostazione di stampa, il codice generato da Excel ha impostato moltissime altre proprietà sempre relative alla stampa.

Questo è un buon esempio di eccesso del registratore macro. Se desiderate una macro che commuti semplicemente l'impostazione della pagina in modo che abbia un orientamento orizzontale, potete semplificare notevolmente questa macro eliminando tutto il codice estraneo. Questo rende la macro un po' più veloce e molto più facile da leggere. Ecco l'aspetto della macro dopo l'eliminazione delle righe non pertinenti:

```
Sub Macro1()
    With ActiveSheet.PageSetup
        .Orientation = xlLandscape
    End With
End Sub
```

L'unica riga necessaria è quella che imposta la proprietà Orientation. In realtà, potete semplificare ancora di più questa macro, perché non avete davvero bisogno del costrutto With-End With (scoprirete ulteriori informazioni su questo costrutto nel Capitolo 14):

```
Sub Macro1()
    ActiveSheet.PageSetup.Orientation = xlLandscape
End Sub
```

In questo caso, la macro modifica solo la proprietà Orientation dell'oggetto PageSetup del foglio attivo (ActiveSheet). Tutte le altre proprietà rimangono invariate. A proposito, `xlLandscape` è una costante VBA che facilita la lettura del codice. Questa costante ha valore `2`, quindi la seguente dichiarazione funziona esattamente allo stesso modo (ma è meno intuitiva):

```
ActiveSheet.PageSetup.Orientation = 2
```

Vi parlerò delle costanti integrate fra poco, nel Capitolo 7.

Anziché registrare questa macro, potete inserirla direttamente in un modulo VBA. Per farlo, dovete sapere quali oggetti, proprietà e metodi usare. Sebbene la macro registrata non sia davvero eccezionale, registrandola vi siete resi conto che l'oggetto PageSetup è contenuto in un oggetto Worksheet e che l'oggetto PageSetup ha una proprietà Orientation. Armati di questa nuova conoscenza e di una rapida incursione nella Guida (e probabilmente di qualche sperimentazione), potrete scrivere la macro manualmente.

Questo capitolo spiega quasi tutto il necessario per utilizzare il registratore di macro. L'unica cosa che vi manca è l'esperienza. Alla fine, scoprirete quali dichiarazioni registrate potete eliminare in tutta sicurezza. Ancora meglio, scoprirete come modificare una macro registrata, per renderla più utile.

3

Concetti di programmazione

IN QUESTA PARTE...

- Scoprire gli elementi essenziali della programmazione VBA in Excel: le variabili, le costanti, i tipi di dati, gli operatori, le matrici e così via.
- Conoscere gli oggetti Range; ne sarete entusiasti.
- Scoprire perché le funzioni VBA (e anche le funzioni del foglio di lavoro di Excel) sono importanti.
- Scoprire l'essenza della programmazione: i processi decisionali e i cicli.
- Eseguire automaticamente il codice in reazione a determinati eventi.
- I diversi tipi di errori. Perché la gestione degli errori è importante?
- Che cosa fare quando del codice buono fa cose cattive: introduzione al debugging.

Capitolo 7

Elementi essenziali del linguaggio VBA

IN QUESTO CAPITOLO

- **Quando, come e perché utilizzare i commenti nel codice**
- **Utilizzare variabili e costanti**
- **Indicare a VBA quale tipo di dati state usando**
- **Familiarizzare con le matrici**
- **Sapere perché potrebbe essere necessario utilizzare le etichette nelle routine**

Poiché VBA è un vero e proprio linguaggio di programmazione, utilizza molti elementi comuni a tutti i linguaggi di programmazione. Questo capitolo introduce molti di questi elementi: i commenti, le variabili, le costanti, i tipi di dati, gli array (o matrici) e alcuni componenti aggiuntivi. Se avete già usato altri linguaggi di programmazione, alcuni di questi concetti vi risulteranno familiari. Se invece siete ai primi passi nella programmazione, allacciatevi le cinture.

Utilizzare i commenti nel codice VBA

Un *commento* è il tipo più semplice di istruzione VBA. Poiché VBA ignora queste istruzioni, potete metterci quello che volete. Potete inserire un commento per ricordarvi perché avete fatto una certa scelta o per chiarire il funzionamento di alcune istruzioni particolarmente complesse che avete scritto.

CONSIGLIO

Usate commenti esaurienti e generosi per descrivere quello che fa il codice (non sempre il suo significato è ovvio, leggendolo). Spesso, il codice che, nella vostra testa ha perfettamente senso oggi, risulterà arcano e misterioso domani. Scriveteli quando siete “sul pezzo”. Non rimandateli a un secondo tempo (mai?).

Iniziate un commento con un apostrofo (`'`). VBA ignora qualsiasi testo che, in una riga di codice, segue un apostrofo. Potete dedicare al commento un’intera riga o potete inserirlo alla fine di una riga di codice. L’esempio seguente mostra una routine VBA con quattro commenti:

```
Sub FormatCells()
'   Esce se non è selezionato un intervallo
     If TypeName(Selection) <> "Range" Then
         MsgBox "Selezionare un intervallo."
         Exit Sub
     End If
'    Formatta le celle
     With Selection
         .HorizontalAlignment = xlRight
         .WrapText = False ' niente a capo
         .MergeCells = False ' nessuna unione
     End With
End Sub
```

RICORDA!

La regola “un apostrofo indica un commento” ha un’eccezione: VBA non interpreta un apostrofo all’interno di una coppia di virgolette come un indicatore di commento. Per esempio, la seguente istruzione non contiene un commento, anche se ha un apostrofo:

```
Msg = "Controllare l'input"
```

Durante la scrittura del codice, potete dover provare a eseguire una routine *escludendo* una particolare istruzione (o un gruppo di istruzioni). Potreste anche eliminare le istruzioni e poi ridigitarle in un secondo momento. Ma sarebbe una perdita di tempo. Una soluzione migliore consiste semplicemente nel trasformare queste istruzioni in commenti, inserendo gli apostrofi. Quando eseguite una routine, VBA ignora le istruzioni che iniziano con apostrofi. Per riattivare le istruzioni “commentate”, basta rimuovere gli apostrofi.

CONSIGLIO

Ecco un modo rapido per convertire un blocco di istruzioni in commenti. In VBE, selezionate Visualizza ➪ Barre degli strumenti ➪ Modifica per visualizzare la barra degli strumenti Modifica. Per convertire un blocco di istruzioni in commenti, selezionate tali istruzioni e fate clic sul pulsante Commento. Per rimuovere gli apostrofi, selezionate le istruzioni e fate clic sul pulsante Rimuovi commento.

Ognuno sviluppa il proprio stile nel redigere i commenti. Per essere utili, tuttavia, i commenti dovrebbero fornire informazioni che non risultano immediatamente evidenti dalla lettura del codice.

I seguenti suggerimenti possono aiutarvi a fare un uso efficace dei commenti.

- Identificatevi come autori. Questo vi sarà utile in seguito a un cambio di mansioni, quando chi vi sostituirà avrà bisogno di porvi delle domande.
- Descrivete brevemente lo scopo di ciascuna routine Sub o Function che scrivete.
- Utilizzate i commenti per tenere traccia anche delle modifiche che avete apportato a una routine.
- Utilizzate un commento per indicare che state utilizzando una funzione o un costrutto in un modo insolito o non standard.
- Utilizzate i commenti per descrivere il significato delle variabili che usate, soprattutto nei casi in cui tali nomi non siano del tutto “parlanti”.
- Utilizzate un commento per descrivere eventuali soluzioni alternative sviluppate per superare i bug in Excel.
- Scrivete commenti mentre sviluppate il codice, invece di rimandarli a un passaggio successivo.
- A seconda del vostro ambiente di lavoro, potreste considerare l’idea di aggiungere un paio di commenti scherzosi. Chi erediterà il vostro posto di lavoro potrebbe apprezzare la battuta.

Variabili, costanti e tipi di dati

Lo scopo principale di VBA è quello di manipolare i dati. VBA memorizza i dati nella memoria del computer, e i dati potrebbero non finire mai su disco. Alcuni dati, come gli intervalli di fogli di lavoro, risiedono all’interno di oggetti. Altri dati sono memorizzati in variabili, create dall’utente.

Che cosa sono le variabili

Una *variabile* è semplicemente una posizione di memoria nel computer, dotata di un nome e utilizzata da un programma. Avete molta flessibilità nel denominare le variabili, quindi fate in modo che il nome delle variabili sia il più descrittivo possibile. Potete assegnare un valore a una variabile utilizzando l’operatore uguale (=). Troverete ulteriori informazioni su questo argomento più avanti nel paragrafo “Le istruzioni di assegnamento”).

Ecco alcuni esempi di variabili alle quali vengono assegnati dei valori. Notate che l’ultimo esempio utilizza due variabili, e non una sola.

```
x = 1
InterestRate = 0.075
LoanPayoffAmount = 243089
DataEntered = False
x = x + 1
UserName = "Bob Johnson"
Date_Started = #3/14/2019#
```

```
MyNum = YourNum * 1.25
```

Il linguaggio VBA ha alcune regole relative ai nomi delle variabili:

- potete utilizzare lettere, numeri e alcuni segni di punteggiatura, ma il primo carattere deve essere una lettera;
- VBA non distingue tra lettere maiuscole e minuscole;
- in un nome di variabile non potete inserire spazi, punti o operatori matematici noti;
- non potete utilizzare i caratteri `#`, `$`, `%`, `&` e `!`;
- i nomi di variabili non possono essere più lunghi di 255 caratteri, ma nessuno si avvicina a quel limite.

Per rendere più leggibili i nomi di variabili, spesso i programmatori utilizzano uno stile "a gobbe di cammello" (come per esempio in InterestRate) o un carattere di sottolineatura (interest_rate).

Il linguaggio VBA offre molte parole riservate, che non potete utilizzare nei nomi di variabili o di routine. Fra queste vi sono parole come Sub, Dim, With, End, Next e For. Se tentate di utilizzare una di queste parole come nome di variabile, riceverete un errore di compilazione (il che significa che il vostro codice non verrà eseguito). Quindi, se un'istruzione di assegnamento genera un messaggio d'errore, ricontrollate che il nome della variabile non sia una parola riservata. Un modo semplice per farlo è selezionare il nome della variabile e premere F1. Se il nome della variabile è una parola riservata, avrà una voce nel sistema Help.

VBA consente di creare variabili con nomi che corrispondono a quelli del modello a oggetti di Excel, come Workbook e Range. Ma ovviamente, usando nomi del genere finirete per confondervi. Ecco per esempio una macro perfettamente valida (ma davvero sconsigliabile) che dichiara Range come nome di variabile e funziona con una cella chiamata Range in un foglio di lavoro denominato Range:

```
Sub RangeConfusion()
     Dim Range As Double
     Range = Sheets("Range").Range("Range").Value
     MsgBox Range
End Sub
```

Resistete quindi all'impulso di usare una variabile chiamata Workbook o Range, e piuttosto usate qualcosa come MyWorkbook o MyRange.

Quali sono i tipi di dati di VBA?

Quando si parla di linguaggi di programmazione, il termine *tipo di dati* fa riferimento al modo in cui il programma memorizza i dati in memoria, per esempio come numeri interi, numeri decimali o stringhe. Sebbene VBA possa occuparsi automaticamente di questi dettagli, lo fa a un costo (non c'è mai niente gratis). Lasciando gestire i dati a VBA si ottiene un'esecuzione più lenta e un uso più inefficiente della memoria. Per piccole applicazioni, questo, di solito, non rappresenta un problema. Ma per applicazioni grandi o complesse, che possono dover operare su grandi quantità di dati o dover economizzare il più possibile in termini di memoria, è necessario avere familiarità con i tipi di dati.

VBA gestisce automaticamente tutti i dettagli dei dati, il che semplifica la vita dei programmatori. Non tutti i linguaggi di programmazione offrono questo lusso. Per esempio, alcuni linguaggi sono *rigorosamente tipizzati*, il che significa che il programmatore deve definire esplicitamente il tipo di dati di ogni variabile utilizzata.

VBA non obbliga a dichiarare le variabili che vengono usate, ma questa è sicuramente una buona pratica. Vedremo il perché più avanti in questo capitolo.

VBA offre una buona varietà di tipi di dati. La Tabella 7.1 ne elenca i più comuni.

TABELLA 7.1 **Tipi di dati standard di VBA**

Tipo di dati	Byte usati	Intervallo di valori
Byte	1	da 0 a 255
Boolean	2	True o False
Integer	2	da -32.768 a 32.767
Long	4	da -2.147.483.648 a 2.147.483.647
Single	4	da -3,40E38 a -1,40E-45 per valori negativi; da 1,40E-45 a 3,40E38 per valori positivi
Double	8	-da 1,79E308 a -4,.94E-324 per valori negativi; da 4,94E-324 a 1,79E308 per valori positivi
Currency	8	da -922.337.203.685.477 a 922.337.203.685.477
Date	8	da 1/1/0100 a 31/12/9999
Object	4	Qualsiasi riferimento a un oggetto
String	1 per carattere	Variabile

In generale, scegliete il tipo di dati che utilizza il minor numero di byte, ma che sia in grado di gestire tutti i valori che dovrete memorizzare nella variabile.

PER I PIÙ CURIOSI

Un'eccezione alla regola "numero minimo di byte" è Long. La maggior parte dei programmatori VBA utilizza Long al posto di Integer perché così facendo si ottiene un leggero incremento di prestazioni. Ma per piccole routine, non noterete alcuna differenza tra i tipi di dati Integer e Long.

Dichiarazione e visibilità delle variabili

Leggendo i paragrafi precedenti, avete fatto la conoscenza con un po' di variabili e di tipi di dati. In questo paragrafo, scoprirete come dichiarate una variabile dicendo che ha le caratteristiche di un determinato tipo di dati.

Se non dichiarate il tipo di dati di una variabile che utilizzate in una routine, VBA utilizza il tipo predefinito: Variant. I dati memorizzati come Variant sono come camaleonti; cambiano tipo in base a quel che ci fate. Per esempio, se una variabile è di tipo Variant e contiene una stringa di testo simile a un numero (come "143"), potete utilizzarla sia per le manipolazioni di stringhe sia per i calcoli numerici. VBA gestisce automaticamente la conversione. Il fatto di lasciare che VBA gestisca i tipi di dati può sembrare una soluzione facile, ma ricordate che in questo modo sacrificate la velocità e aumentate la memoria utilizzata.

Prima di utilizzare una variabile in una routine, è un'ottima idea quella di *dichiararla*, ovvero di dire a VBA il tipo di dati che pensate di memorizzarvi. La dichiarazione delle variabili accelera la macro e le consente di utilizzare la memoria in modo più efficiente. Il tipo di dati predefinito, Variant, fa sì che VBA esegua ripetutamente lunghe verifiche e riservi più memoria del necessario. Se invece VBA conosce il tipo di dati della variabile, non dovrà più immaginare nulla e potrà riservarle la quantità di memoria sufficiente per archiviare i dati.

Per auto-obbligarvi a dichiarare tutte le variabili che usate, includete nel vostro modulo VBA, queste due parole riservate come prima istruzione:

```
Option Explicit
```

Con questa istruzione, non sarete più in grado di eseguire il codice contenente variabili che non avete preventivamente dichiarato.

RICORDA!

Basta specificare Option Explicit una sola volta: all'inizio del modulo, prima della dichiarazione di eventuali routine. Tenete presente che l'istruzione Option Explicit si applica solo al modulo in cui si trova. Se un progetto è costituito da più moduli VBA, dovrete inserire un'istruzione Option Explicit in ciascun modulo.

Supponete di utilizzare una variabile non dichiarata (ovvero una Variant) denominata CurrentRate. A un certo punto della vostra routine, inserite la seguente istruzione:

```
CurentRate =.075
```

Il nome della variabile è errato (manca una r), cosa che può essere molto difficile da individuare. Se non noterete l'errore, Excel lo interpreterà come *un'altra variabile* e produrrà quasi immancabilmente risultati errati. Se ponete Option Explicit all'inizio del modulo (obbligandovi di fatto a dichiarare la variabile CurrentRate), Excel genererà un errore quando incontrerà una variante errata di tale variabile.

CONSIGLIO

Per garantire che l'istruzione Option Explicit venga usata automaticamente ogni volta che inserite un nuovo modulo VBA, attivate l'opzione Dichiarazione di variabili obbligatoria. La trovate nella scheda Editor della finestra di dialogo Opzioni (in VBE, scegliete Strumenti ➪ Opzioni).

CONSIGLIO

La dichiarazione delle variabili consente inoltre di sfruttare una scorciatoia che permette di risparmiare un po' di digitazione. Basta digitare i primi due o tre caratteri del nome della variabile e premere Ctrl + barra spaziatrice. VBE completerà la voce per voi o, se la scelta è ambigua, vi mostrerà un elenco di parole corrispondenti, tra le quali potrete selezionare quella corretta. In realtà, questo trucco funziona anche con parole e funzioni riservate. La Figura 7.1 mostra un esempio del suo funzionamento.

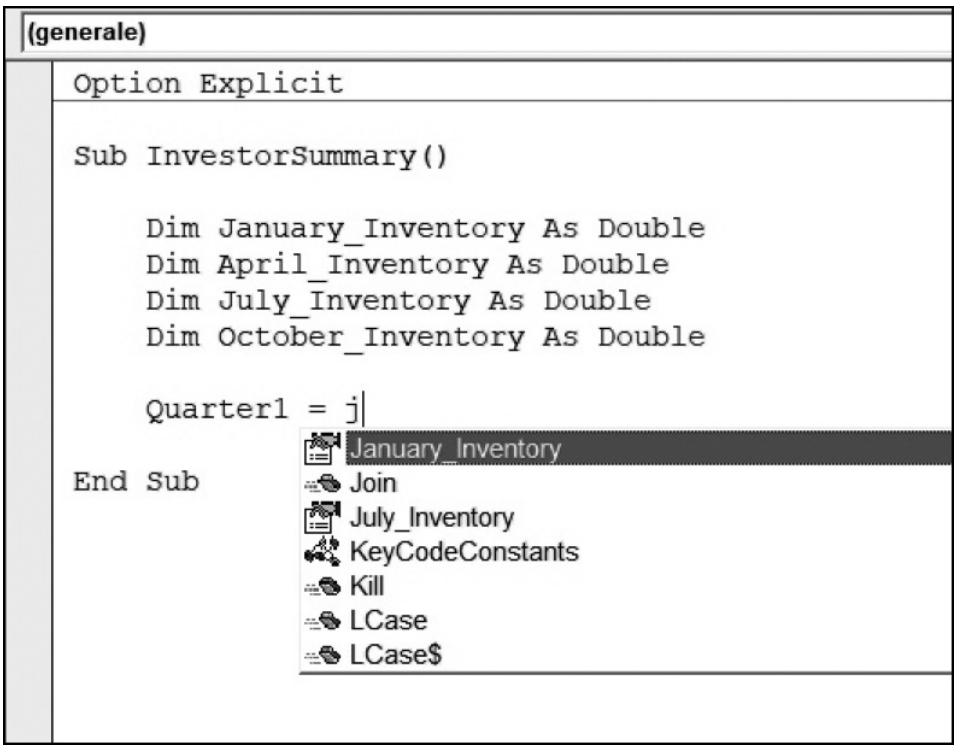

FIGURA 7.1 Premendo Ctrl + barra spaziatrice otterrete un elenco di nomi di variabili, parole riservate e funzioni.

Ora conoscete i vantaggi insiti nel dichiarare le variabili. Ma *come si fa*? Il modo più comune consiste nell'usare un'istruzione Dim. Ecco alcuni esempi di dichiarazioni di variabili:

```
Dim YourName As String
Dim January_Inventory As Double
Dim AmountDue As Double
Dim RowNumber As Long
Dim X
```

Le prime quattro variabili sono dichiarate di un tipo di dati specifico. L'ultima variabile, X, è dichiarata senza un tipo di dati specifico, quindi viene considerata Variant (può essere qualsiasi cosa).

Oltre a Dim, VBA ha tre parole riservate che vengono utilizzate per dichiarare le variabili:

- Static;
- Public;
- Private.

Parlerò più avanti delle parole riservate Dim, Static, Public e Private, ma in primo luogo, a questo punto sono rilevanti altri due concetti: quello di visibilità e di vita di una variabile.

Una cartella di lavoro può avere un numero qualsiasi di moduli VBA e un modulo VBA può avere numero qualsiasi di routine Sub e Function. La *visibilità* di una variabile determina quali moduli e routine possono utilizzarla. La Tabella 7.2 contiene i dettagli di questo concetto.

TABELLA 7.2 **Visibilità della variabile**

Livello di visibilità	Come viene dichiarata la variabile
Solo routine	Usando un'istruzione Dim o un'istruzione Static nella routine che utilizza la variabile.
Solo modulo	Usando un'istruzione Dim o Private che preceda la prima istruzione Sub o Function nel modulo.

Tutte le routine in tutti i moduli	Usando un'istruzione Public prima della prima istruzione Sub o Function di un modulo.

Confusi, vero? Continuate a leggere; alcuni esempi chiariranno meglio questi concetti.

Variabili per una sola routine

Il livello di utilizzo più basso di visibilità di una variabile è a livello di routine (una *routine* è una Sub o una Function). Le variabili dichiarate con questo livello di visibilità possono essere utilizzate solo nella routine in cui sono state dichiarate. Al termine della routine, la variabile non esiste più ed Excel libera la sua area di memoria. Se eseguite nuovamente la routine, la variabile viene ricreata, e il suo valore precedente viene perso.

Il modo più comune per dichiarare una variabile per la sola routine è con un'istruzione Dim. Dim è un vecchio termine di programmazione che è l'abbreviazione di *dimensione*, il che significa semplicemente che state mettendo da parte della memoria (di una certa dimensione) per una particolare variabile. Di solito, le istruzioni Dim vengono posizionate immediatamente dopo l'istruzione Sub o Function e prima del codice della routine.

Nell'esempio seguente vengono mostrate alcune variabili per la sola routine, dichiarate utilizzando le istruzioni Dim:

```
Sub MySub()
     Dim x As Integer
     Dim First As Long
     Dim InterestRate As Single
     Dim TodaysDate As Date
     Dim UserName As String
     Dim MyValue
'   ...      [Qui va il codice della routine]  ...
End Sub
```

Notate che l'ultima istruzione Dim nell'esempio precedente non dichiara il tipo di dati della variabile MyValue; dichiara solo la variabile stessa. L'effetto è che la variabile MyValue sarà di tipo Variant.

WARNING

A differenza di alcuni linguaggi, VBA non consente di dichiarare un gruppo di variabili come un particolare tipo di dati, separando le variabili con virgole. Per esempio, sebbene valida, la seguente istruzione *non* dichiara tutte le variabili (i, j e k) come Integer:

```
Dim i, j, k As Integer
```

In questo esempio, solo k viene dichiarata come un numero intero; le altre variabili saranno di tipo Variant.

Se dichiarate una variabile con livello di visibilità pari alla sola routine, le altre routine dello stesso modulo potranno utilizzare una loro variabile con lo stesso nome, ma ciascuna istanza della variabile sarà specifica della relativa routine. In generale, le variabili dichiarate a livello di routine sono più efficienti perché al termine della routine VBA libera la memoria che esse usano.

Variabili per un solo modulo

A volte c'è bisogno di una variabile che sia disponibile per tutte le routine di un modulo. In tal caso, basta dichiarare la variabile (utilizzando Dim o Private) *prima* della prima istruzione Sub o Function del modulo, e dunque all'esterno di ogni routine. Lo si fa nella sezione Dichiarazioni, all'inizio del modulo (lo stesso luogo in cui si trova l'istruzione Option Explicit).

La Figura 7.2 mostra come sapere quando ci si trova nella sezione Dichiarazioni. Utilizzate il menu a discesa a destra e potrete andare direttamente alla sezione Dichiarazioni. E… no, non passate dal Via e non prendete i 500 euro.

Supponete di voler dichiarare la variabile CurrentValue in modo che sia disponibile per tutte le routine nel modulo. Tutto quello che dovete fare è usare l'istruzione Dim nella sezione Dichiarazioni:

```
Dim CurrentValue As Double
```

Con questa dichiarazione nella posizione corretta, la variabile Current Value può essere utilizzata da qualsiasi altra routine all'interno del modulo e manterrà il suo valore da una routine all'altra.

FIGURA 7.2 Ogni modulo VBA ha una sezione Dichiarazioni, che appare prima di qualsiasi routine Sub o Function.

Variabili pubbliche

Se dovete rendere disponibile una variabile per tutte le routine di tutti i moduli VBA di una cartella di lavoro, dichiaratela a livello di modulo (nella sezione Dichiarazioni) utilizzando la parola riservata Public. Ecco un esempio:

```
Public CurrentRate As Long
```

La parola riservata Public rende la variabile CurrentRate disponibile in qualsiasi routine della cartella di lavoro, anche quelle che si trovano in altri moduli VBA. È necessario inserire questa istruzione prima della prima istruzione Sub o Function di un modulo.

PER I PIÙ CURIOSI

Se desiderate che una variabile sia disponibile per i moduli anche di altre cartelle di lavoro, dovete dichiararla Public e stabilire un riferimento alla cartella di lavoro che contiene la dichiarazione della variabile. Potete impostare un riferimento scegliendo in VBE il comando Strumenti ➪ Riferimenti. In pratica, la condivisione di una variabile tra più cartelle di lavoro non viene quasi mai sfruttata. Ma è bello sapere che si può fare. Consideratela pura cultura personale.

Variabili statiche

Normalmente, al termine di una routine, tutte le variabili della routine vengono eliminate. Le *variabili statiche* sono casi speciali, perché mantengono il proprio valore anche al termine della routine. Dichiarate una variabile statica a livello di routine. Una variabile statica può essere utile se dovete tenere traccia del numero di volte in cui viene eseguita una routine. Potete dichiarare una variabile statica e incrementarla a ogni esecuzione della routine.

Come mostrato nel seguente esempio, potete dichiarare variabili statiche usando la parola riservata Static:

```
Sub MySub()
     Static Counter As Integer
     Dim Msg As String
     Counter = Counter + 1
     Msg = "Numero di esecuzioni: " & Counter
     MsgBox Msg
End Sub
```

Il codice tiene traccia del numero di volte in cui la routine è stata eseguita e visualizza il numero in una finestra. Il valore della variabile Counter non viene cancellato al termine della routine, ma solo quando si chiude e poi si riapre la cartella di lavoro.

WARNING

Anche se il valore di una variabile statica viene mantenuto dopo la fine di una variabile, tale variabile non sarà disponibile per le altre routine. Nell’esempio della routine MySub precedente, la variabile Counter e il suo valore sono disponibili solo all’interno della routine MySub. In altre parole, è una variabile la cui visibilità non esce dalla routine.

Vita delle variabili

Nulla vive per sempre, nemmeno le variabili. Il livello di visibilità di una variabile non solo determina dove può essere usata quella variabile, ma influenza anche le circostanze in cui la variabile viene eliminata.

Potete eliminare tutte le variabili dalla memoria in tre modi.

- Facendo clic sul pulsante Ripristina, nella barra degli strumenti (il piccolo pulsante quadrato blu nella barra degli strumenti Standard di VBE).
- Facendo clic su Fine nella finestra di dialogo del messaggio d'errore durante l'esecuzione di una routine.
- Includendo un'istruzione End in qualsiasi punto del codice. Questa non equivale a un'istruzione End Sub o End Function.

Altrimenti, quando il codice della macro avrà completato l'esecuzione, verranno rimosse dalla memoria tutte le variabili dichiarate a livello delle routine. Le variabili statiche, quelle dichiarate a livello del modulo e le variabili globali (pubbliche) mantengono tutte il loro valore da un'esecuzione all'altra del codice.

WARNING

Se utilizzate variabili a livello di modulo o globali, assicuratevi che abbiano i valori che vi aspettate. Non sapete mai se una delle situazioni appena menzionate potrebbe aver causato la perdita del contenuto di tali variabili!

Le costanti

Il valore di una variabile può cambiare (e di solito cambia), durante l'esecuzione della routine. Non a caso si chiama *variabile*. A volte, è necessario far riferimento a un valore o una stringa che non cambia mai. In tal caso, avete bisogno di una *costante* – un elemento dotato di nome il cui valore non cambia mai.

Come mostrato nei seguenti esempi, potete dichiarare delle costanti usando l'istruzione Const. L'istruzione di dichiarazione assegna anche un valore alla costante:

```
Const NumQuarters As Integer = 4
Const Rate = .0725, Period = 12
Const ModName As String = "Budget Macros"
Public Const AppName As String = "Budget Application"
```

CONSIGLIO

L'utilizzo di costanti al posto di valori o stringhe costituisce un'ottima pratica di programmazione. Se la routine deve far riferimento più volte a un valore ben preciso (per esempio un tasso di interesse), è meglio dichiararlo come una costante e poi far riferimento a tale nome anziché al valore. Ciò rende il codice più leggibile e più facile da modificare. E se il tasso di interesse dovesse cambiare, vi basterà cambiare una sola dichiarazione.

RICORDA!

Come le variabili, anche le costanti hanno un livello di visibilità. Tenete a mente questi concetti.

- Per rendere disponibile una costante solo all'interno di una singola routine, dichiaratela dopo l'istruzione Sub o Function della routine.
- Per rendere disponibile una costante per tutte le routine di un modulo, dichiaratela nella sezione Dichiarazioni del modulo.
- Per rendere disponibile una costante per tutti i moduli della cartella di lavoro, utilizzate la parola riservata Public e dichiaratela nella sezione Dichiarazioni di un qualsiasi modulo.

A differenza di una variabile, il valore di una costante non cambia mai. Se tentaste di modificare il valore di una costante in una routine VBA, otterrete un errore. Questo non è troppo sorprendente, perché è abbastanza ovvio che il valore di una costante debba rimanere… costante. Se vi capita di dover modificare il valore di una costante durante l'esecuzione del codice, allora avete bisogno di una variabile, non di una costante.

Costanti predefinite

Excel e VBA impiegano molte costanti predefinite, che potete utilizzare senza doverle dichiarare. Il registratore di macro utilizza in genere costanti anziché valori effettivi. In generale, per utilizzarle, non è necessario conoscerne il valore. La seguente semplice routine utilizza una costante predefinita (xlCalculation Manual) per modificare la proprietà Calculation dell'oggetto Application (in altre parole, per impostare la modalità di ricalcolo manuale di Excel):

```
Sub CalcManual()
     Application.Calculation = xlCalculationManual
End Sub
```

Se cercate nella Guida di Excel le modalità di calcolo, troverete quanto segue:

Nome	Valore	Descrizione
xlCalculationAutomatic	-4105	Excel controlla il ricalcolo.
xlCalculationManual	-4135	Il calcolo viene eseguito solo quando l'utente lo richiede.
xlCalculationSemiautomatic	2	Excel controlla il ricalcolo, ma ignora le modifiche nelle tabelle.

Quindi il valore effettivo della costante predefinita xlCalculationManual è -4135. Ovviamente, è più facile usare il nome della costante che cercare di ricordare un valore così strano. Come potete vedere, molte delle costanti predefinite sono numeri arbitrari, che tuttavia hanno un significato speciale per VBA.

CONSIGLIO

Per trovare il valore effettivo di una costante predefinita, utilizzate la finestra Immediata di VBE ed eseguite un'istruzione VBA come la seguente:

```
? xlCalculationAutomatic
```

Se la finestra Immediata non è visibile, premete Ctrl + G. Il punto interrogativo è una scorciatoia per non dover digitare **Print**.

Lavorare con le stringhe

Excel è in grado di gestire numeri e testi, quindi non dovrebbe sorprendere che VBA abbia questa stessa capacità. I testi vengono spesso chiamati *stringhe*. In VBA potete lavorare con due tipi di stringhe.

- **Stringhe di lunghezza fissa**, dichiarate con un numero specifico di caratteri. La lunghezza massima è 65.526 caratteri. Sono parecchi! Per farvi un'idea, questo capitolo contiene circa la metà di quei caratteri.
- **Stringhe di lunghezza variabile**, che teoricamente possono contenere fino a due miliardi di caratteri. Se riusciste a digitare 5 caratteri al secondo, impieghereste circa 760 giorni per produrre due miliardi di caratteri, senza mai mangiare o dormire.

Quando dichiarate una variabile stringa con un'istruzione Dim, potete specificare la lunghezza massima se la conoscete (e sarà una stringa di lunghezza fissa) o lasciare che sia VBA a gestirla dinamicamente (e sarà una stringa di lunghezza variabile). L'esempio seguente dichiara la variabile MyString come una stringa della lunghezza massima di 50 caratteri (l'uso dell'asterisco serve a specificare il numero di caratteri, fino al limite di 65.526 caratteri). Anche YourString viene dichiarata come una stringa, ma la sua lunghezza non è specificata:

```
Dim MyString As String * 50
Dim YourString As String
```

WARNING

Quando dichiarate una stringa di una lunghezza fissa superiore a 999 caratteri, non utilizzate alcun separatore delle migliaia nel numero che specifica la dimensione della stringa. In effetti, non utilizzate mai i separatori delle migliaia nei valori numerici, in VBA: non gli piacciono.

Lavorare con le date

Un altro tipo di dati che potreste trovare utile è Date. Per memorizzare le date potreste utilizzare anche variabili stringa, ma poi non potreste eseguire calcoli fra date. L'uso del tipo Date offre alle vostre routine maggiore flessibilità.

Per esempio, potreste dover calcolare il numero di giorni trascorsi tra due date. Questo sarebbe impossibile (o almeno estremamente impegnativo) usando due stringhe per contenere le date.

Una variabile definita come Date può contenere date che vanno dal 1 gennaio 0100 al 31 dicembre 9999. Si tratta di un intervallo di quasi 10.000 anni ed è più che sufficiente anche per le previsioni finanziarie più ardite. Potete utilizzare il tipo di dati Date anche per lavorare con i dati orari (a VBA manca un tipo di dati Time).

Questi esempi dichiarano variabili e costanti di tipo Date:

```
Dim Today As Date
Dim StartTime As Date
Const FirstDay As Date = #1/1/2019#
Const Noon = #12:00:00#
```

In VBA, dovete inserire le date e le ore tra due segni di hash (cancelletto), come mostrato negli esempi precedenti.

RICORDA!

Le variabili di tipo Date visualizzano le date in base al formato "data breve" del sistema e visualizzano le ore in base al formato dell'ora del sistema (formattazione a 12 o 24 ore). Windows memorizza queste impostazioni e potete modificarle tramite la finestra di dialogo Area geografica nel Pannello di controllo di Windows. Pertanto, il formato di data o ora visualizzato in VBA può variare in base alle impostazioni del sistema su cui l'applicazione è in esecuzione.

Quando scrivete del codice VBA, tuttavia, dovrete impiegare un formato per date americano (come mm/gg/aaaa). Pertanto, la seguente istruzione assegna un giorno di *ottobre* (e non di novembre) alla variabile MyDate (anche se il vostro sistema è impostato per visualizzare le date in formato gg/mm/aaaa):

```
MyDate = #10/11/2019#
```

Quando poi visualizzerete la variabile (con la funzione MsgBox, per esempio), VBA mostrerà MyDate utilizzando le impostazioni di sistema. Pertanto, se il sistema utilizza il formato gg/mm/aaaa, MyDate verrà visualizzata come 11/10/2019. Non è bellissimo, ma dovrete farci l'abitudine: è il prezzo da pagare per poter usare le routine anche al di fuori dei confini nazionali.

Le istruzioni di assegnamento

Un'*istruzione di assegnamento* è un'istruzione VBA che assegna il risultato di un'espressione a una variabile o a un oggetto. La Guida di Excel definisce il termine espressione come:

> Una combinazione di parole riservate, operatori, variabili e costanti che produce una stringa, un numero o un oggetto. Un'espressione può essere usata per eseguire un calcolo, manipolare caratteri o eseguire test sui dati.

Gran parte del vostro lavoro in VBA comporterà lo sviluppo (e il debugging) di *espressioni*. Se sapete creare formule in Excel, non avrete problemi a creare espressioni. Con una formula del foglio di lavoro, Excel visualizza il risultato in una cella. Un'espressione VBA, d'altra parte, può essere assegnata a una variabile.

Esempi di istruzioni di assegnamento

Negli esempi di istruzioni di assegnamento che seguono, le espressioni sono la parte a destra del segno di uguale:

```
x = 1
x = x + 1
x = (y * 2) / (z * 2)
HouseCost = 375000
FileOpen = True
Range("TheYear").Value = 2019
```

CONSIGLIO

Le espressioni possono essere complesse quanto necessarie; per rendere più leggibili le espressioni lunghe, usate il carattere di continuazione della riga (uno spazio seguito da un carattere di sottolineatura).

Spesso, le espressioni utilizzano funzioni: funzioni standard VBA, funzioni del foglio di lavoro Excel o funzioni sviluppate con VBA. Parlerò di funzioni nel Capitolo 9.

A proposito del segno di uguale

Come potete vedere dall'esempio precedente, VBA usa il segno di uguale come operatore di assegnamento. Probabilmente siete abituati a usare il segno di uguale come simbolo matematico per l'uguaglianza. Pertanto, una istruzione di assegnamento come la seguente potrebbe farvi alzare le sopracciglia:

```
z = z + 1
```

In quale pazzo universo z è uguale a se stesso più 1? Risposta: in nessun universo conosciuto. In questo caso, l'istruzione di assegnamento (una volta eseguita) aumenta di 1 unità il valore di z. Quindi se z è 12, l'esecuzione dell'istruzione rende z uguale a 13. Ricordate che un'assegnamento usa il segno di uguale come operatore, non come simbolo di uguaglianza.

Operatori vari

In VBA, gli operatori ricoprono ruoli importanti. Oltre all'operatore di uguale (discusso nel paragrafo precedente), VBA offre diversi operatori. Li trovate elencati nella Tabella 7.3. Questi dovrebbero esservi familiari, perché sono gli stessi operatori utilizzati nelle formule del foglio di lavoro (tranne l'operatore Mod).

TABELLA 7.3 **Operatori VBA**

Funzione	Simbolo dell'operatore
Addizione	+
Moltiplicazione	*
Divisione	/
Sottrazione	-
Elevamento a potenza	^
Concatenamento di stringhe	&
Divisione intera (il risultato è sempre un intero)	\
Modulo aritmetico (restituisce il resto di un'operazione di divisione)	Mod

Quando si scrive una formula in Excel, si esegue il modulo aritmetico utilizzando la funzione MOD. Per esempio, la seguente formula restituisce 2 (il resto della divisione di 12 per 5):

```
=MOD(12,5)
```

In VBA, l'operatore Mod viene usato in questo modo (e alla fine z vale 2):

```
z = 12 Mod 5
```

PER I PIÙ CURIOSI

Il termine *concatenare* in programmazione significa "unire insieme". Quindi, se si concatenano due stringhe, le si combinano per creare una nuova stringa.

Come mostrato nella Tabella 7.4, VBA fornisce anche un insieme completo di operatori logici.

TABELLA 7.4 **Operatori logici di VBA**

Operatore	Uso
Not	Esegue una negazione logica su un'espressione
And	Esegue una congiunzione logica su due espressioni
Or	Esegue una disgiunzione logica su due espressioni
Xor	Esegue un'esclusione logica su due espressioni
Eqv	Esegue un'equivalenza logica su due espressioni
Imp	Esegue un'implicazione logica su due espressioni

L'ordine di precedenza per gli operatori in VBA è esattamente lo stesso delle formule di Excel. L'esponenziazione ha la precedenza più elevata. Dopo vengono le moltiplicazioni e le divisioni, seguite dalle addizioni e sottrazioni. Potete usare le parentesi per modificare l'ordine di precedenza, facendo in modo che qualsiasi operazione racchiusa tra parentesi venga prima di qualsiasi operatore. Date un'occhiata a questo codice:

```
x = 3
y = 2
z = x + 5 * y
```

Quando viene eseguito il codice precedente, qual è il valore di z? Se avete risposto 13, meritate una stella d'oro che dimostra che avete compreso il concetto di precedenza degli operatori. Se avete risposto 16, vi dico che l'operazione di moltiplicazione (5 * y) viene sempre eseguita per prima, e poi al risultato viene aggiunta x.

Se avete difficoltà a ricordare la corretta precedenza dell'operatore, aggiungete le parentesi nelle parti che vengono calcolate per prime. Per esempio, la precedente istruzione di assegnamento, con le parentesi, si presenta così:

```
z = x + (5 * y)
```

CONSIGLIO

Non abbiate timore nell'usare le parentesi anche quando non sarebbero obbligatorie, specialmente se, così facendo, il codice diventa più facile da comprendere. VBA non se ne avrà a male se userete delle parentesi di troppo.

Lavorare con le matrici: gli array

Come molti linguaggi di programmazione, VBA supporta il concetto di array, che in italiano chiama "matrice". Una *matrice* è un gruppo di *più variabili* che condividono *un unico nome*. Potete far riferimento a una variabile specifica della matrice utilizzando il nome della matrice e un numero di indice, indicato tra parentesi. Per esempio, potete definire una matrice di 12 variabili stringa per contenere i nomi dei mesi dell'anno. Se assegnate alla matrice il nome *MonthNames*, potrete far riferimento al primo elemento della matrice come MonthNames (1), al secondo elemento come MonthNames (2) e così via.

Dichiarare le matrici

Prima di poter usare una matrice, occorre dichiararla. Nessuna eccezione. A differenza delle normali variabili, VBA è molto severo su questa regola. Potete dichiarare una matrice con un'istruzione Dim, proprio come si fa per una comune variabile.

Tuttavia, è necessario anche specificare il numero di elementi della matrice. Potete farlo specificando il primo indice, la parola riservata To e l'ultimo indice, tutti tra parentesi. L'esempio seguente dichiara una matrice di 100 numeri interi:

```
Dim MyArray(1 To 100) As Integer
```

Quando dichiarate una matrice, potete scegliere di specificare anche solo l'indice superiore. Se omettete l'indice inferiore, VBA presuppone che sia 0. Pertanto, le seguenti dichiarazioni dichiarano entrambe la stessa matrice di 101 elementi:

```
Dim MyArray (0 To 100) As Integer
Dim MyArray (100) As Integer
```

CONSIGLIO

Se desiderate che VBA stabilisca che l'indice inferiore per le matrici sia 1 (anziché lo standard, 0), includete, nella sezione Dichiarazioni, ovvero nella parte superiore del modulo, la seguente istruzione:

```
Option Base 1
```

Questa istruzione impone a VBA di utilizzare 1 come primo indice per tutte le matrici che dichiarano solo l'indice superiore. Con questa istruzione presente, le seguenti istruzioni sono identiche ed entrambe dichiarano una matrice di 100 elementi:

```
Dim MyArray (1 To 100) As Integer
Dim MyArray (100) As Integer
```

Matrici multidimensionali

Le matrici create negli esempi precedenti sono tutte monodimensionali. Potete considerare le matrici monodimensionali come un'unica fila di valori. Le matrici che create in VBA possono avere fino a 60 dimensioni, anche se raramente avrete bisogno di matrici di più di due o tre dimensioni. L'esempio seguente dichiara una matrice bidimensionale di 81 interi:

```
Dim MyArray (1 To 9, 1 To 9) As Integer
```

Questa è una matrice di 9 × 9 elementi, perfetta per ospitare tutti i numeri del Sudoku.

Per far riferimento a un determinato elemento di questa matrice, basta specificare due numeri indice (in pratica la "riga" e la "colonna" della matrice). L'esempio seguente mostra come assegnare un valore a un elemento di questa matrice:

```
MyArray (3, 4)= 125
```

Questa istruzione assegna un valore a un singolo elemento nella matrice. Considerando che si tratta di una matrice 9 × 9, questa istruzione assegna il numero 125 all'elemento situato nella terza riga e quarta colonna della matrice.

Ecco invece come si dichiara una matrice tridimensionale di 1.000 elementi:

```
Dim My3DArray (1 To 10, 1 To 10, 1 To 10) As Integer
```

Potete pensare a una matrice tridimensionale come a un cubo. Visualizzare matrici di più di tre dimensioni è più difficile.

Matrici dinamiche

Potete anche creare *matrici dinamiche*. Una matrice dinamica non ha un numero di elementi preimpostato. Potete dichiarare una matrice dinamica con una serie vuota di parentesi:

```
Dim MyArray () As Integer
```

Prima di poter utilizzare questa matrice, dovrete utilizzare l'istruzione ReDim per comunicare a VBA quanti elementi contiene. Di solito, il numero di elementi della matrice viene determinato mentre il codice è in esecuzione. Potete utilizzare l'istruzione ReDim un numero qualsiasi di volte, modificando le dimensioni della matrice tutte le volte che volete. L'esempio seguente mostra come si modifica il numero di elementi di una matrice dinamica. Si presuppone che la variabile NumElements contenga un valore calcolato dal codice.

```
ReDim MyArray (1 To NumElements)
```

WARNING

Quando ridimensionate una matrice usando ReDim, cancellate tutti i valori attualmente memorizzati negli elementi della matrice. Potete evitare di distruggere i vecchi valori usando la parola riservata Preserve. L'esempio seguente mostra come preservare i valori di una matrice anche dopo un ridimensionamento:

```
ReDim Preserve MyArray (1 To NumElements)
```

Se MyArray dispone attualmente di dieci elementi ed eseguite l'istruzione precedente con NumElements uguale a 12, i primi dieci elementi rimarranno al loro posto e la matrice avrà spazio per due elementi in più (fino al numero contenuto nella variabile NumElements). Se NumElement fosse uguale a 7, invece, i primi sette elementi verranno mantenuti, ma i restanti tre andranno incontro alla loro fine.

L'argomento delle matrici verrà affrontato di nuovo nel Capitolo 10, quando esploreremo il concetto di ciclo.

Le etichette

Nelle prime versioni di BASIC, ogni riga di codice richiedeva un numero di riga. Se, per esempio, avessi scritto un programma BASIC negli anni Settanta (con i canonici jeans a campana), avrei prodotto qualcosa del genere:

```
010: LET X=5
020: LET Y=3
030: LET Z=X*Y
040: PRINT Z
050: END
```

RICORDA!

VBA consente l’uso di numeri di riga e perfino di etichette testuali. In genere non si utilizza un’etichetta per ogni riga, ma è possibile che occasionalmente sia necessario utilizzarne una. Per esempio si usa un’etichetta per un’istruzione GoTo (discussa nel Capitolo 10). Un’etichetta deve partire dal primo carattere di una riga e terminare con un carattere di due punti.

Capitolo 8

Gli oggetti Range

IN QUESTO CAPITOLO

- **Perché gli oggetti Range sono così importanti**
- **I tanti modi per far riferimento agli intervalli**
- **Le proprietà più utili dell'oggetto Range**
- **I metodi più utili dell'oggetto Range**

Questo capitolo scava un po' più a fondo nei meandri di Excel e dà uno sguardo più da vicino agli oggetti Range. Excel è interamente basato su celle e l'oggetto Range è un contenitore per celle. Perché avete bisogno di sapere così tanto sugli oggetti Range? Perché gran parte del lavoro di programmazione che svolgerete in Excel si concentrerà sugli oggetti Range.

Una rapida occhiata

Un *oggetto Range* rappresenta un intervallo contenuto in un oggetto Worksheet. Gli oggetti Range, come tutti gli altri oggetti, hanno proprietà (che potete esaminare e talvolta modificare) e metodi (che eseguono azioni sull'oggetto).

Un oggetto Range può essere piccolo come una singola cella (per esempio, B4) o grande come tutte le 17.179.869.184 celle di un foglio di lavoro (A1:XFD1048576).

Quando fate riferimento a un oggetto Range, l'indirizzo è sempre racchiuso da virgolette doppie, come in questo caso:

```
Range("A1:C5")
```

Ma anche se l'intervallo è costituito da una sola cella, avete comunque bisogno delle virgolette:

```
Range("K9")
```

Se l'intervallo ha un nome (creato con Formule ➪ Nomi definiti ➪ Definisci nome), potete far riferimento all'intervallo in base al suo nome (sempre racchiuso tra virgolette):

```
Range("PriceList")
```

WARNING

A meno che non indichiate diversamente a Excel, qualificando il riferimento all'intervallo, si presume che stiate facendo riferimento a un intervallo del foglio di lavoro attivo. Se è attivo qualcosa di diverso da un foglio di lavoro (per esempio un foglio grafico), il riferimento all'intervallo non funzionerà, la macro visualizzerà un messaggio d'errore e poi si interromperà bruscamente.

Come mostrato nell'esempio seguente, potete far riferimento a un intervallo che si trova al di fuori del foglio attivo, qualificando il riferimento con il nome del foglio di lavoro della cartella di lavoro attiva:

```
Worksheets("Foglio1").Range("A1:C5")
```

Se poi dovete far riferimento a un intervallo di un'altra cartella di lavoro (ovvero qualsiasi cartella di lavoro diversa da quella attiva), potete utilizzare un'istruzione come questa:

```
Workbooks("Budg.xlsx").Worksheets("Foglio1").Range("A1:C5")
```

Un oggetto Range può essere costituito da una o più righe o colonne, intere. Potete far riferimento a un'intera riga (in questo caso, la riga 3) usando la seguente sintassi:

```
Range("3:3")
```

Potete far riferimento a un'intera colonna (la quarta, D, in questo esempio) in questo modo:

```
Range("D:D")
```

In Excel, potete selezionare intervalli non contigui tenendo premuto Ctrl mentre selezionate vari intervalli con il mouse. La Figura 8.1 mostra una selezione di un intervallo non contiguo. Non vi sorprenderà sapere che VBA consente anche di lavorare con intervalli non contigui. La seguente espressione fa riferimento a un intervallo non contiguo composto da due aree. Notate che le due aree sono separate da una virgola.

```
Range("A1:B8,D9:G16")
```

FIGURA 8.1 Selezione di un intervallo non contiguo.

WARNING

Notate che alcuni metodi e proprietà non gradiscono troppo gli intervalli non contigui. Potreste dover elaborare ogni area separatamente, utilizzando un ciclo.

Altri modi per far riferimento a un intervallo

Più lavorerete con VBA, più vi accorgerete di come sia un linguaggio ben concepito e, di solito, abbastanza logico (nonostante quello che potreste pensare di lui in questo momento). Spesso VBA offre più modi per eseguire un'azione e poi potrete scegliere la soluzione più appropriata per il vostro problema. In questo paragrafo vedremo alcuni altri modi per far riferimento a un intervallo.

PER I PIÙ CURIOSI

Questo capitolo scalfisce appena la superficie delle proprietà e dei metodi dell'oggetto Range. Lavorando con VBA, avrete probabilmente bisogno di accedere ad altre proprietà e metodi. La Guida è il luogo migliore per studiarli, ma è anche una buona idea registrare le azioni e poi esaminare il codice generato da Excel. Probabilmente vi sarete stancati di ascoltare questo consiglio, ma è davvero un buon consiglio.

La proprietà Cells

Invece di utilizzare la parola riservata Range, potete far riferimento a un intervallo tramite la proprietà Cells.

PER I PIÙ CURIOSI

Sebbene le celle possano sembrare un oggetto (o una collezione), in realtà non lo sono. Cells è piuttosto una proprietà valutata da VBA. Il linguaggio VBA restituisce poi un oggetto (in particolare, un oggetto Range). Se questo vi suona strano, non vi preoccupate. Anche Microsoft sembra in confusione su questo argomento. In alcune versioni precedenti di Excel, la proprietà Cells era chiamata metodo Cells. Indipendentemente da cosa sia, basta capire che Cells è un modo pratico per far riferimento a un intervallo.

La proprietà Cells accetta due argomenti: un numero di riga e un numero di colonna. Entrambi sono argomenti numerici, anche se di solito si fa riferimento alle colonne usando le lettere. La seguente espressione, per esempio, fa riferimento alla cella C2 del Foglio2:

```
Worksheets("Foglio2").Cells(2, 3)
```

Inoltre potete utilizzare la proprietà Cells anche per far riferimento a un intervallo multicella. Il seguente esempio mostra la sintassi da utilizzare:

```
Range(Cells(1, 1), Cells(10, 8)
```

Questa espressione fa riferimento a un intervallo di 80 celle che si estende dalla cella A1 (riga 1, colonna 1) alla cella H10 (riga 10, colonna 8).

Le seguenti istruzioni producono entrambe lo stesso risultato; inseriscono il valore 99 in un intervallo di celle 10 per 8. Per la precisione, queste istruzioni impostano la proprietà Value dell'oggetto Range:

```
Range("A1:H10").Value = 99
Range(Cells(1, 1), Cells(10, 8)).Value = 99
```

CONSIGLIO

Il vantaggio dell'utilizzo della proprietà Cells per far riferimento agli intervalli diventa evidente quando si utilizzano, come argomenti delle celle, variabili anziché numeri. E le cose si chiariranno ulteriormente quando comprenderete il concetto di ciclo nel Capitolo 10.

La proprietà Offset

La proprietà Offset fornisce un altro utile strumento per far riferimento agli intervalli. Questa proprietà, che opera su un oggetto Range e restituisce un altro oggetto Range, consente di far riferimento a una cella che si trova a un certo numero di righe e colonne di distanza da un'altra cella.

Come la proprietà Cells, anche la proprietà Offset accetta due argomenti. Il primo rappresenta il numero di righe di distanza e il secondo il numero di colonne di distanza.

La seguente espressione fa riferimento alla cella che si trova una riga sotto la cella A1 e due colonne a destra della cella A1. In altre parole, fa riferimento alla cella C2:

```
Range("A1").Offset(1, 2)
```

La proprietà Offset può anche utilizzare argomenti *negativi*. Un offset di riga negativo fa riferimento a una riga sopra l'intervallo. Un offset di colonna negativo fa riferimento a una colonna a sinistra dell'intervallo. Il seguente esempio fa riferimento alla cella A1:

```
Range("C2").Offset(-1, -2)
```

E come potete immaginare, potete usare lo zero per uno o entrambi gli argomenti di Offset. La seguente espressione fa riferimento alla cella A1:

```
Range("A1").Offset(0, 0)
```

Ecco un'istruzione che inserisce l'ora del giorno nella cella immediatamente a destra della cella attiva:

```
ActiveCell.Offset(0,1) = Time
```

Quando registrate una macro in modalità relativa, Excel utilizza proprio la proprietà Offset. Per un esempio, fate riferimento al Capitolo 6.

CONSIGLIO

La proprietà Offset è molto utile quando, per gli argomenti, si usano variabili piuttosto che valori effettivi. Il Capitolo 10 include vari esempi in questo senso.

Alcune proprietà utili dell'oggetto Range

Un oggetto Range è dotato di decine di proprietà. Potreste scrivere programmi VBA ininterrottamente per un anno intero e non usarle mai tutte. Questo paragrafo descrive brevemente alcune delle proprietà più comunemente utilizzate. Per i dettagli, consultate la Guida di VBE.

RICORDA!

Alcune proprietà di Range sono di *sola lettura*, il che significa che il codice può visualizzare il loro valore ma non può modificarlo ("Guardare e non toccare"). Per esempio, ogni oggetto Range ha una proprietà Address, che contiene l'indirizzo dell'intervallo. Potete accedere a questa proprietà di sola lettura, ma non potete cambiarla, il che, pensandoci, ha perfettamente senso.

Gli esempi che seguono, comunque, in genere sono istruzioni e non routine complete. Se volete provarne qualcuna (e dovreste), create una routine Sub. Inoltre, molte di queste istruzioni funzionano correttamente solo se il foglio attivo è un foglio di lavoro.

La proprietà Value

La proprietà Value rappresenta il valore contenuto in una cella. È una proprietà di lettura e scrittura, quindi il codice VBA può leggerla, ma anche modificarne il valore.

La seguente istruzione apre una finestra che mostra il valore contenuto nella cella A1 di Foglio1:

```
MsgBox Worksheets("Foglio1").Range("A1").Value
```

È ovvio che potete leggere la proprietà Value solo per un oggetto Range a cella singola. La seguente istruzione, per esempio, genera un errore:

```
MsgBox Worksheets("Foglio1").Range("A1:C3").Value
```

Tuttavia, potete modificare la proprietà Value per un intervallo di qualsia si dimensione. La seguente istruzione inserisce il numero 123 in ogni cella dell'intervallo:

```
Worksheets("Foglio1").Range("A1:C3").Value = 123
```

CONSIGLIO

Value è anche la *proprietà predefinita* per un oggetto Range. In altre parole, se omettete la proprietà per un intervallo, Excel utilizzerà la proprietà Value. Le seguenti istruzioni inseriscono entrambe il valore 75 nella cella A1 del foglio di lavoro attivo:

```
Range("A1").Value = 75
Range("A1") = 75
```

PER I PIÙ CURIOSI

ASSEGNARE I VALORI DI UN INTERVALLO MULTICELLA A UNA VARIABILE

Anche se potete leggere la proprietà Value solo per un oggetto Range a cella singola, potete assegnare a una variabile i valori in un intervallo multicella, purché la variabile sia di tipo Variant. Questo perché un Variant può comportarsi come una matrice. Ecco un esempio:

```
Dim x As Variant
x = Range("A1:C3").Value
```

Quindi potete trattare la variabile x come se fosse una matrice. Questa istruzione, per esempio, restituisce il valore contenuto nella cella B1:

```
MsgBox x(1, 2)
```

La proprietà Text

La proprietà Text restituisce una stringa che rappresenta il testo così come viene visualizzato in una cella: il valore formattato. La proprietà Text è di sola lettura. Supponiamo che la cella A1 contenga il valore 12.3 e sia formattata per visualizzare due decimali e un segno di dollaro ($ 12,30). La seguente istruzione mostra una finestra contenente $ 12.30:

```
MsgBox Worksheets("Foglio1").Range("A1").Text
```

Ma la prossima istruzione mostra una finestra contenente il valore 12.3:

```
MsgBox Worksheets("Foglio1").Range("A1").Value
```

Se la cella contiene una formula, la proprietà Text restituisce il suo risultato. Se la cella contiene del testo, la proprietà Text e la proprietà Value restituiscono lo stesso valore, poiché un testo (a differenza di un numero) non può essere formattato per essere visualizzato in modo diverso.

La proprietà Count

La proprietà Count restituisce il numero di celle contenute in un intervallo. Conta tutte le celle, non solo le celle “piene”. Count è una proprietà di sola lettura, come probabilmente immaginate. La seguente istruzione accede alla proprietà Count di un intervallo e visualizza il risultato (9) in una finestra:

```
MsgBox Range("A1:C3").Count
```

Le proprietà Column e Row

La proprietà Column restituisce il numero di colonna di un intervallo monocella. Sua sorella, la proprietà Row, restituisce il numero di riga di un intervallo monocella. Entrambe sono proprietà di sola lettura. La seguente istruzione, per esempio, visualizza 6, perché la cella F3 si trova nella sesta colonna:

```
MsgBox Sheets("Foglio1").Range("F3").Column
```

L’espressione successiva visualizza 3 perché la cella F3 si trova nella terza riga:

```
MsgBox Sheets("Foglio1").Range("F3").Row
```

RICORDA!

Se l’oggetto Range è costituito da più celle, la proprietà Column restituisce il numero corrispondente alla prima colonna dell’intervallo e analogamente la proprietà Row restituisce il numero corrispondente alla prima riga

dell'intervallo.

Non confondete le proprietà Column e Row con le proprietà Columns e Rows (discusse in precedenza in questo capitolo). Le proprietà Column e Row restituiscono un singolo valore. Le proprietà Columns e Rows, al contrario, restituiscono un oggetto Range. Che differenza fa una semplice "s" in più...

La proprietà Address

Address, una proprietà di sola lettura, visualizza l'indirizzo di una cella di un oggetto Range come riferimento assoluto (con un segno di dollaro prima della lettera della colonna e prima del numero della riga). La seguente istruzione mostra la finestra rappresentata nella Figura 8.2:

```
MsgBox Range(Cells(1, 1), Cells(5, 5)).Address
```

La proprietà HasFormula

La proprietà HasFormula (che è di sola lettura) restituisce True se l'intervallo monocella contiene una formula. Restituisce False se la cella contiene qualcosa di diverso da una formula (o è vuota). Se l'intervallo è composto da più di una cella, VBA restituisce True solo se tutte le celle dell'intervallo contengono una formula e False solo se nessuna delle celle dell'intervallo contiene una formula. Restituisce Null se l'intervallo contiene una combinazione di formule e non-formule. Null è una specie di terra di nessuno: la risposta non è né vera né falsa, e qualsiasi cella dell'intervallo può avere o meno una formula.

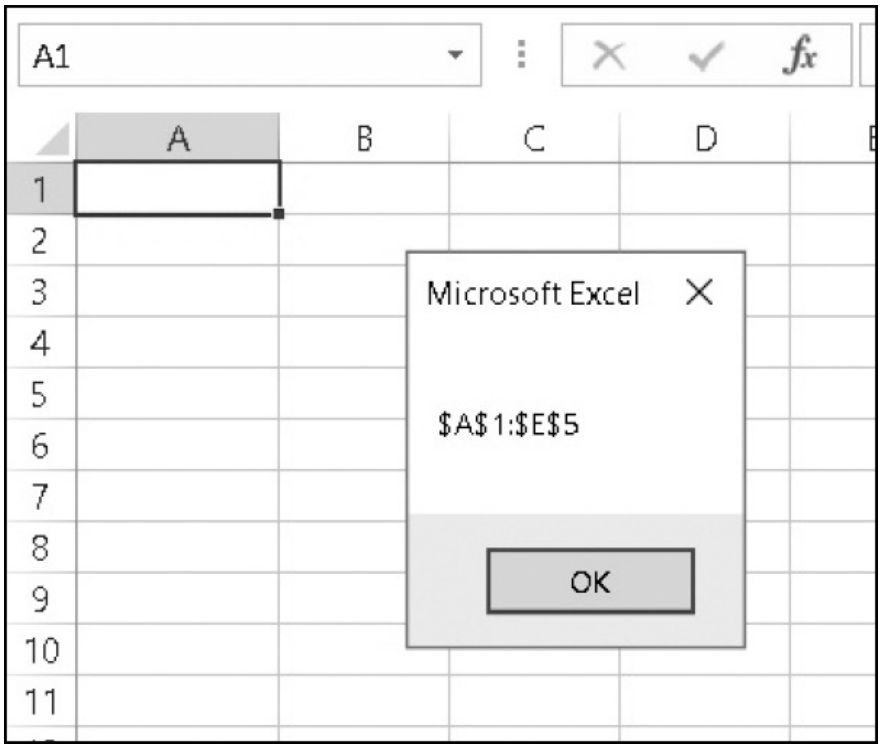

FIGURA 8.2 Questa finestra visualizza la proprietà Address di un intervallo 5x5.

WARNING

È necessario fare attenzione quando si lavora con proprietà che possono restituire Null. Più in particolare, l'unico tipo di dati che può gestire Null è Variant.

Supponete, per esempio, che la cella A1 contenga un valore e che la cella A2 contenga una formula. Le seguenti istruzioni generano un errore, perché l'intervallo non è costituito solo da formule o da non-formule:

```
Dim FormulaTest As Boolean
FormulaTest = Range("A1:A2").HasFormula
```

Il tipo di dati Boolean può gestire solo i valori True o False. Null fa sì che Excel si lamenti e visualizzi un messaggio d'errore. Per risolvere questo tipo di situazione, la cosa migliore da fare è assicurarsi che la variabile FormulaTest sia dichiarata come Variant e non come Boolean. L'esempio seguente utilizza la comoda funzione TypeName di VBA (insieme a un costrutto If-Then-Else) per determinare il tipo di dati della variabile FormulaTest. Se l'intervallo ha una combinazione di formule e non formule, la finestra visualizza *Misto!*. Altrimenti visualizza True o False.

```
Sub CheckForFormulas()
     Dim FormulaTest As Variant
     FormulaTest = Range("A1:A2").HasFormula
     If TypeName(FormulaTest) = "Null" Then
         MsgBox "Misto!"
     Else
         MsgBox FormulaTest
     End If
End Sub
```

Per ulteriori informazioni sull'uso del costrutto If-Then-Else consultate il Capitolo 10.

La proprietà Font

Come ho detto in precedenza in questo capitolo (vedi "La proprietà Cells"), una proprietà può restituire un oggetto. La proprietà Font di un oggetto Range è un altro esempio dell'uso di questo concetto. La proprietà Font restituisce un oggetto Font.

Come potreste aspettarvi, un oggetto Font ha molte proprietà. Per modificare alcuni aspetti del tipo di carattere di un intervallo, è necessario innanzitutto accedere all'oggetto Font dell'intervallo e poi manipolarne le proprietà. Sì, la cosa è un po' confusa lo ammetto, ma forse questo esempio vi sarà d'aiuto.

La seguente istruzione utilizza la proprietà Font dell'oggetto Range per restituire un oggetto Font. Poi imposta a True la proprietà Bold dell'oggetto Font. In parole povere, questa istruzione visualizza la cella in grassetto:

```
Range("A1").Font.Bold = True
```

La verità è che non avete davvero bisogno di sapere che state utilizzando un oggetto Font contenuto in un oggetto Range. Finché utilizzate la sintassi corretta, tutto funziona perfettamente. Spesso, registrando le azioni, il registratore di macro vi dice tutto ciò che dovete sapere sulla sintassi corretta.

Per ulteriori informazioni sulla registrazione di macro consultate il Capitolo 6.

La proprietà Interior

Ecco ancora un altro esempio di proprietà che restituisce un oggetto. La proprietà Interior di un oggetto Range restituisce un oggetto Interior (sì, è un nome un po' macabro, ma non è colpa sua). Questo tipo di riferimento a oggetti funziona allo stesso modo della proprietà Font descritta nel paragrafo precedente.

La seguente istruzione, per esempio, modifica la proprietà Color dell'oggetto Interior contenuta nell'oggetto Range:

```
Range("A1").Interior.Color = 8421504
```

In altre parole, questa istruzione modifica lo sfondo della cella, applicando un grigio medio. Come? Non sapevate che 8421504 è il grigio medio? Per approfondimenti sul meraviglioso mondo dei colori di Excel, consultate il riquadro "Quattro parole sui colori".

QUATTRO PAROLE SUI COLORI

Prima di Excel 2007, Microsoft ha cercato di convincerci che in un foglio di lavoro 56 colori erano sufficienti. Le cose sono cambiate, e oggi in una cartella di lavoro possiamo usare oltre 16 milioni di colori: 16.777.216 colori, per la precisione.

Molti oggetti hanno una proprietà Color, che accetta valori che vanno da 0 a 16777215. Nessuno, credo, può ricordare così tanti valori, quindi (fortunatamente) esiste un modo più semplice per specificarli: la funzione RGB di VBA. Questa funzione sfrutta il fatto che potete rappresentare uno qualsiasi di questi sedici milioni di colori con vari livelli di rosso, verde e blu. I tre argomenti nella funzione RGB corrispondono ai componenti rosso, verde e blu del colore e ognuno di questi argomenti può essere compreso tra 0 e 255.

Ovviamente 256 x 256 x 256 = 16.777.216: il numero di colori disponibili. È bello quando la matematica funziona alla perfezione!

Ecco un'istruzione che cambia il colore di sfondo di una cella in un colore del tutto casuale:

```
Range("A1").Interior.Color = Int(16777216 * Rnd)
```

Di seguito sono riportati alcuni esempi che utilizzano la funzione RGB per modificare il colore di sfondo di una cella:

```
Range("A1").Interior.Color = RGB(0, 0, 0) ' nero
Range("A1").Interior.Color = RGB(255, 0, 0) ' rosso
Range("A1").Interior.Color = RGB(0, 0, 255) ' blu
Range("A1").Interior.Color = RGB(198, 212, 60) ' un verde
Range("A1").Interior.Color = RGB(128, 128, 128) ' grigio medio
```

Qual è il valore esatto di RGB (128, 128, 128)? Questa istruzione vi dice che è 8421504:

```
MsgBox RGB(128, 128, 128)
```

Se volete usare i colori standard, potete scegliere di utilizzare una delle costanti standard di VBA: vbBlack, vbRed, vbGreen, vbYellow, vbBlue, vbMagenta, vbCyan o vbWhite. Per esempio, la seguente istruzione rende gialla la cella A1:

```
Range("A1").Interior.Color = vbYellow
```

Excel 2007 ha inoltre introdotto i *colori dei temi*. Sono i colori visualizzati quando utilizzate, per esempio, il controllo Colore di riempimento, nel gruppo Carattere della scheda Home. Provate a registrare una macro e a modificare i colori, e otterrete qualcosa di simile a:

```
Range("A1").Interior.ThemeColor = xlThemeColorAccent4
Range("A1").Interior.TintAndShade = 0.399975585192419
```

Sì, abbiamo da conoscere altre due proprietà correlate al colore. ThemeColor (il colore di base, specificato come una costante predefinita), più un valore TintAndShade, che rappresenta quanto è scuro o chiaro il colore. I valori di TintAndShade vanno da -1.0 a +1.0. I valori positivi rendono il colore più chiaro, mentre quelli negativi lo rendono più scuro. Quando impostate un colore utilizzando la proprietà ThemeColor, il colore cambia quando si applica un tema diverso (selezionando il comando Layout di pagina ➪ Temi ➪ Temi).

La proprietà Formula

La proprietà Formula rappresenta la formula contenuta in una cella. È una proprietà di lettura-scrittura, quindi potete usarla per visualizzare la formula contenuta in una cella o anche per inserire una nuova formula in una cella. Per esempio, la seguente istruzione inserisce una formula nella cella A13:

```
Range("A13").Formula = "=SUM(A1:A12)"
```

Notate che la formula è una stringa di testo ed è racchiusa tra virgolette. Notate inoltre che la formula inizia con un segno di uguale, come tutte le formule.

RICORDA!

Notate anche come per inserire nella cella la funzione SOMMA di Excel dovete usare il suo nome originale inglese: SUM. Mettetela così: VBA è possessivo. Non gli va che siate voi a tradurre le funzioni di Excel, pretende di farlo lui. Potete anche scrivere sum in lettere minuscole e non batterà ciglio, ma non provatevi a tradurre la funzione in italiano. L'aspetto positivo? Il solito: le vostre routine funzioneranno sempre in tutto il mondo.

Se la formula contiene delle virgolette, le cose si fanno un po' complicate. Immaginate di voler inserire questa formula usando VBA:

```
=SUM(A1:A12)&" punti vendita"
```

Questa formula visualizza un valore seguito da "punti vendita". Per rendere accettabile questa formula, è necessario raddoppiare le virgolette nella formula. Altrimenti, VBA si confonderà e si lamenterà (e a ragione!) di un errore di sintassi. Quindi, ecco un'istruzione che inserisce nella cella una formula che contiene virgolette:

```
Range("A13").Formula = "=SUM(A1:A12)&"" punti vendita"""
```

A proposito, potete accedere alla proprietà Formula di una cella anche se la cella non contiene una formula. Se una cella non contiene una formula, la proprietà Formula restituisce la stessa cosa della proprietà Value.

Se volete sapere se una cella contiene una formula, utilizzate la proprietà HasFormula (discussa in precedenza in questo capitolo).

La proprietà NumberFormat

La proprietà NumberFormat rappresenta il formato numerico (espresso come stringa di testo) dell'oggetto Range. È una proprietà di lettura-scrittura, quindi il codice VBA può esaminare il formato del numero o anche modificarlo. La seguente istruzione cambia il formato numerico della colonna A nel formato percentuale con due cifre decimali:

```
Columns("A:A").NumberFormat = "0.00%"
```

Ecco come visualizzare un elenco degli altri formati numerici (meglio ancora, attivate contemporaneamente il registratore di macro).

1. **Attivate un foglio di lavoro.**
2. **Premete Ctrl + 1 per accedere alla finestra di dialogo Formato celle.**
3. **Fate clic sulla scheda Numero.**
4. **Selezionate la categoria Personalizzato per visualizzare e applicare alcune stringhe di formattazione dei numeri.**

Alcuni metodi utili dell'oggetto Range

Come sapete, in VBA un metodo esegue un'azione. Un oggetto Range ha decine di metodi ma, di nuovo, non è davvero il caso di conoscerli tutti. In questo paragrafo, esplorerete alcuni dei metodi più utili dell'oggetto Range.

Il metodo Select

Utilizzate il metodo Select per selezionare un intervallo di celle. La seguente istruzione seleziona un intervallo nel foglio di lavoro attivo:

```
Range("A1:C12").Select
```

WARNING

Prima di selezionare un intervallo, utilizzate un'istruzione aggiuntiva per assicurarvi che sia attivo il foglio di lavoro corretto. Per esempio, se Foglio1 contiene l'intervallo che desiderate selezionare, utilizzate le seguenti istruzioni per selezionare l'intervallo:

```
Sheets("Foglio1").Activate
Range("A1:C12").Select
```

Contrariamente a quanto potreste aspettarvi, l'istruzione seguente genera un errore se Foglio1 non è già il foglio attivo. In altre parole, è necessario utilizzare due istruzioni invece di una: una per attivare il foglio e un'altra per selezionare l'intervallo.

```
Sheets("Foglio1").Range("A1:C12").Select
```

CONSIGLIO

Se utilizzate il metodo GoTo dell'oggetto Application per selezionare un intervallo, potete evitare di selezionare il foglio di lavoro corretto. Questa istruzione attiva Foglio1 e poi seleziona l'intervallo:

```
Application.Goto Sheets("Foglio1").Range("A1:C12")
```

Il metodo GoTo è l'equivalente VBA della pressione del tasto F5 in Excel, che visualizza la finestra di dialogo Vai a.

I metodi Copy e Paste

In VBA potete eseguire le operazioni di copia e incolla utilizzando i metodi Copy e Paste. Notate come entrino in gioco due oggetti diversi. Il metodo Copy è applicabile all'oggetto Range, ma il metodo Paste si applica all'oggetto Worksheet. In realtà la cosa ha senso: copiate un intervallo, ma poi lo incollate in un foglio di lavoro.

Questa breve macro (per gentile concessione del registratore di macro) copia l'intervallo A1:A12 e lo incolla nello stesso foglio di lavoro, a partire dalla cella C1:

```
Sub CopyRange()
     Range("A1:A12").Select
     Selection.Copy
     Range("C1").Select
     ActiveSheet.Paste
End Sub
```

CONSIGLIO

Notate che nell'esempio precedente, viene utilizzato il metodo Paste dell'oggetto ActiveSheet. Questa è una versione speciale dell'oggetto Worksheet che fa riferimento al foglio di lavoro attualmente attivo. Notate inoltre che la macro seleziona l'intervallo prima di copiarlo. Tuttavia, non è necessario selezionare un intervallo prima di svolgerci un'azione. La seguente routine, in effetti, esegue lo stesso compito dell'esempio precedente, ma con una singola istruzione:

```
Sub CopyRange2()
     Range("A1:A12").Copy Range("C1")
End Sub
```

Questa routine sfrutta il fatto che il metodo Copy può utilizzare un argomento che corrisponde all'intervallo di destinazione per l'operazione di copia. È qualcosa che potete scoprire controllando la Guida.

Il metodo Clear

Il metodo Clear cancella il contenuto di un intervallo, oltre a tutta la formattazione della cella. Se, per esempio, volete cancellare tutto il contenuto nella colonna D, ecco l'istruzione da usare:

```
Columns("D:D").Clear
```

Dovreste fare la conoscenza di altri due metodi correlati. Il metodo ClearContents elimina il contenuto dell'intervallo, ma lascia intatta la formattazione. Il metodo ClearFormats elimina la formattazione dell'intervallo, ma non il contenuto delle celle.

Il metodo Delete

In termini di cancellazione di un intervallo, Clear si differenzia da Delete. Quando eliminate (Delete) un intervallo, Excel sposta le celle rimanenti per chiudere la "ferita".

Nell'esempio seguente il metodo Delete elimina la riga 6:

```
Rows("6:6").Delete
```

Quando eliminate (Delete) un intervallo che non è un'intera riga o colonna, Excel deve sapere come spostare le celle (per vederne il funzionamento, provate in Excel il comando Home ➪ Celle ➪ Elimina ➪ Elimina celle).

La seguente istruzione cancella un intervallo e riempie il vuoto prodottosi spostando le altre celle a sinistra:

```
Range("C6:C10").Delete xlToLeft
```

Il metodo Delete utilizza un argomento che indica a Excel come spostare le celle rimanenti. In questo caso, come argomento, viene utilizzata la costante predefinita (xlToLeft).

Capitolo 9

Usare VBA e le funzioni del foglio di lavoro

IN QUESTO CAPITOLO

- **Usare le funzioni per rendere più potenti le espressioni VBA**
- **Le funzioni integrate di VBA**
- **Uso delle funzioni del foglio di lavoro di Excel nel codice VBA**
- **Scrivere funzioni personalizzate**

Esistono tre tipi di funzioni: quelle integrate in VBA, quelle integrate in Excel e altre funzioni scritte in VBA. I capitoli precedenti spiegano come utilizzare le funzioni nelle espressioni VBA e questo capitolo completa tale spiegazione. Le funzioni possono fare in modo che il codice VBA esegua alcune azioni davvero potenti con uno sforzo di programmazione minimo o nullo. Se questa idea vi ha fatto alzare le antenne, questo capitolo fa per voi.

Che cos'è una funzione?

Tranne poche persone che trattano Excel come una specie di elaboratore di testi, tutti gli utenti di Excel incorporano, nelle loro formule, le varie funzioni del foglio di lavoro. La funzione più comune è SOMMA ma ne avete oltre cinquecento a disposizione.

In buona sostanza una funzione esegue un calcolo e restituisce un valore. La funzione SOMMA, ovviamente, restituisce la somma di un intervallo di valori. Lo stesso vale per le funzioni utilizzate nelle espressioni VBA: ogni funzione esegue un compito e restituisce un valore.

Le funzioni che usate in VBA possono essere di tre tipi:

- funzioni integrate, fornite da VBA;
- funzioni del foglio di lavoro, fornite da Excel;
- funzioni personalizzate che voi (o qualcun altro) scrivete, usando VBA. Il resto di questo capitolo chiarisce le differenze.

Usare le funzioni integrate di VBA

VBA offre numerose funzioni integrate. Alcune di queste accettano argomenti, altre no.

Esempi di funzioni VBA

Questo paragrafo presenta alcuni esempi di utilizzo delle funzioni VBA nel codice. In molti di questi esempi, la funzione MsgBox visualizza un valore in una finestra. Sì, MsgBox è una funzione VBA piuttosto insolita, ma è pur sempre una funzione. Questa utile funzione visualizza un messaggio in una finestra di dialogo e restituisce un valore. Per ulteriori dettagli sulla funzione MsgBox, consultate il Capitolo 15.

CONSIGLIO

Sul sito web di questo libro trovate una cartella di lavoro che contiene tutti gli esempi.

Visualizzare la data e l’ora del sistema

Il primo esempio utilizza la funzione VBA Date per visualizzare in una finestra la data di sistema:

```
Sub ShowDate()
     MsgBox "Oggi è: " & Date
End Sub
```

Notate che la funzione Date non usa un argomento. A differenza delle funzioni del foglio di lavoro, una funzione VBA senza argomenti non richiede una coppia di parentesi, vuota. Se anche digitate una coppia di parentesi vuota, il VBE le rimuove.

Per ottenere l’ora, utilizzate la funzione Time. Se desiderate ottenerle entrambe, usate la funzione Now, che restituisce sia la data sia l’ora.

Trovare la lunghezza di una stringa

La seguente routine utilizza la funzione Len, che restituisce la lunghezza di una stringa di testo. La funzione Len accetta un argomento: la stringa. Quando eseguite questa routine, nella finestra verrà visualizzato il vostro nome e il numero di caratteri che occupa (vedere la Figura 9.1).

```
Sub GetLength()
     Dim MyName As String
     Dim StringLength As Long
     MyName = Application.UserName
     StringLength = Len(MyName)
     MsgBox MyName & " ha " & StringLength & " caratteri."
End Sub
```

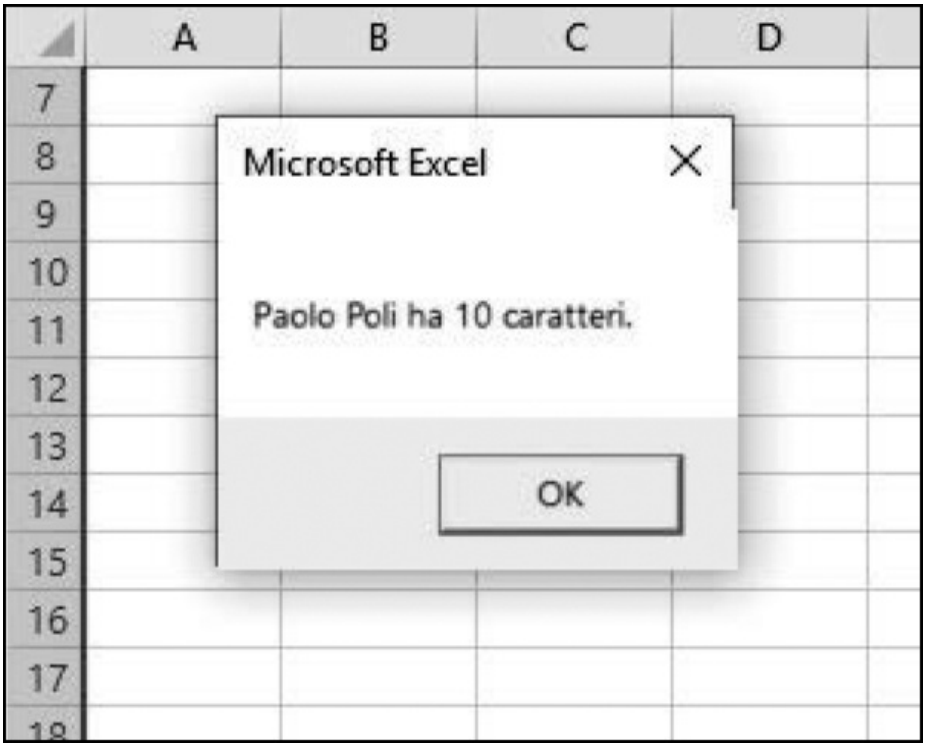

FIGURA 9.1 Calcola la lunghezza del vostro nome.

RICORDA!

Excel ha anche una funzione LEN (LUNGHEZZA), che potete usare nelle formule del vostro foglio di lavoro. La versione (inglese) di Excel e la funzione VBA funzionano allo stesso modo.

Visualizzare il nome di un mese

La seguente routine utilizza la funzione MonthName, che restituisce il nome di un mese. MonthName utilizza un argomento: un numero intero compreso tra 1 e 12.

```
Sub ShowMonthName()
     Dim ThisMonth As Long
     ThisMonth = Month(Date)
     MsgBox MonthName(ThisMonth)
End Sub
```

Questa routine utilizza la funzione Month per ottenere il mese corrente (come valore) e questo valore viene assegnato alla variabile ThisMonth. La funzione MonthName converte poi il valore in testo. Se eseguite questa routine in aprile, nella finestra del messaggio viene visualizzato il testo “aprile”.

In realtà, la variabile ThisMonth non è necessaria. Potete ottenere lo stesso risultato anche con questa espressione, che utilizza tre funzioni VBA:

```
MonthName(Month(Date))
```

Qui, la data corrente viene passata come argomento alla funzione Month, che restituisce un valore passato come argomento alla funzione MonthName.

Determinare la dimensione di un file

La seguente routine Sub visualizza le dimensioni, in byte, del file eseguibile di Excel. Ottenete questo valore utilizzando la funzione FileLen:

```
Sub GetFileSize()
     Dim TheFile As String
     TheFile = "C:\Program Files (x86)\Microsoft Office\root\Office16\EXCEL.EXE"
     MsgBox FileLen(TheFile)
End Sub
```

Notate che questa routine *specifica* il nome file (ovvero, ne indica esplicitamente il percorso). Questa non è mai una buona idea. Il file potrebbe non essere sull’unità C o nella cartella Excel e potrebbe avere un altro nome. La seguente istruzione mostra un approccio migliore:

```
TheFile = Application.Path & "\EXCEL.EXE"
```

Path è una proprietà dell’oggetto Application. Restituisce semplicemente il nome della cartella in cui è installata l’applicazione, ovvero Excel.

Identificare il tipo dell’oggetto selezionato

La seguente routine utilizza la funzione TypeName, che restituisce il tipo dell’oggetto selezionato nel foglio di lavoro (come stringa):

```
Sub ShowSelectionType()
     Dim SelType As String
     SelType = TypeName(Selection)
     MsgBox SelType
End Sub
```

La selezione potrebbe essere un intervallo, un’immagine, un rettangolo, un grafico o qualsiasi altro tipo di oggetto che può essere selezionato.

CONSIGLIO

La funzione TypeName è molto versatile. Potete utilizzarla anche per determinare il tipo di dati di una variabile.

Funzioni VBA che fanno qualcosa di più che restituire un valore

Alcune funzioni VBA vanno ben oltre il loro dovere. Oltre a restituire un semplice valore, hanno alcuni effetti collaterali utili. La Tabella 9.1 li elenca.

TABELLA 9.1 **Funzioni VBA con effetti collaterali utili**

Funzione	Che cosa fa
MsgBox	Visualizza una comoda finestra di dialogo contenente un messaggio e dei pulsanti. La funzione restituisce un codice che identifica il pulsante scelto dall’utente. Per ulteriori dettagli vedete il Capitolo 15.

InputBox	Visualizza una semplice finestra di dialogo che richiede all'utente alcuni input. La funzione restituisce qualunque cosa l'utente inserisca nella finestra di dialogo. Il Capitolo 15 esplora la funzione InputBox.
Shell	Esegue un altro programma. La funzione restituisce il *task ID* (un identificativo univoco) dell'altro programma (o un errore se la funzione non può avviarlo).

Conoscere le funzioni VBA

Come scoprire quali funzioni offre VBA? Bella domanda. La migliore fonte è la Guida a VBA. Un altro modo consiste nel digitare **VBA** seguito da un punto. Otterrete un elenco di elementi, come illustrato nella Figura 9.2. Quelli con un'icona verde sono funzioni. Se questo trucco non funziona, selezionate Strumenti ➪ Opzioni, fate clic sulla scheda Editor e selezionate il segno di spunta accanto a Elenco membri automatico.

FIGURA 9.2 Un modo per visualizzare l'elenco delle funzioni VBA.

VBA offre oltre 140 diverse funzioni. Alcune sono così specializzate e arcane che non ne avrete mai bisogno. Altre, tuttavia, vi saranno utili in molte applicazioni. La Tabella 9.2 elenca alcune delle funzioni più utili.

TABELLA 9.2 **Le funzioni standard più utili di VBA**

Funzione	Che cosa fa
`Abs`	Restituisce il valore assoluto di un numero.
`Array`	Restituisce un Variant contenente una matrice.
`Choose`	Restituisce un valore tratto da un elenco di elementi.
`Chr`	Converte un valore ANSI in una stringa.
`CurDir`	Restituisce il percorso corrente.
`Data`	Restituisce la data corrente.
`DateAdd`	Restituisce una data cui è stato aggiunto un intervallo di tempo specificato, per esempio un mese da una data specifica.
`DateDiff`	Restituisce un numero intero che mostra il numero di intervalli di tempo specificati tra due date: per esempio, il numero di mesi tra oggi e il giorno del vostro compleanno.
`DatePart`	Restituisce un intero contenente la parte specificata di una determinata data, per esempio un dato giorno dell'anno.
`DateSerial`	Converte una data in un numero seriale.
`DateValue`	Converte una stringa in una data.
`Day`	Restituisce il giorno del mese da una data.
`Dir`	Restituisce il nome di un file o di una directory che corrisponde a un modello.
`Err`	Restituisce il codice d'errore in una condizione di errore.
`Error`	Restituisce il messaggio d'errore che corrisponde a un codice d'errore.
`Exp`	Restituisce la base del logaritmo naturale (e) elevato a una potenza.
`FileLen`	Restituisce il numero di byte in un file.
`Fix`	Restituisce la parte intera di un numero.
`Format`	Visualizza un'espressione in un formato particolare.

GetSetting	Restituisce un valore tratto dal registro di Windows.
Hour	Restituisce la parte oraria di un'ora.
InputBox	Visualizza una finestra di input.
InStr	Restituisce la posizione di una stringa all'interno di un'altra stringa (partendo dall'inizio).
InStrRev	Restituisce la posizione di una stringa all'interno di un'altra stringa (partendo dalla fine).
Int	Restituisce la parte intera di un numero.
IsArray	Restituisce True se la variabile è una matrice.
IsDate	Restituisce True se l'espressione è una data.
IsEmpty	Restituisce True se la variabile non è stata inizializzata.
IsError	Restituisce True se l'espressione è un valore di errore.
IsMissing	Restituisce True se a una routine non è stato passato un argomento facoltativo.
IsNull	Restituisce True se l'espressione non contiene dati validi.
IsNumeric	Restituisce True se l'espressione può essere valutata come un numero.
LBound	Restituisce l'indice più piccolo per una dimensione di una matrice.
LCase	Restituisce una stringa convertita in lettere minuscole.
Left	Restituisce un numero specificato di caratteri tratti dalla sinistra di una stringa.
Len	Restituisce il numero di caratteri contenuti in una stringa.
Mid	Restituisce un numero specificato di caratteri tratti da una stringa.
Minute	Restituisce la parte dei minuti di un valore orario.
Month	Restituisce il mese da un valore di data.
MsgBox	Apre una finestra di messaggio e (facoltativamente) restituisce un valore.
Now	Restituisce la data e l'ora correnti del sistema.
Replace	Sostituisce una sottostringa in una stringa con un'altra sottostringa.
RGB	Restituisce un valore numerico RGB che rappresenta un colore.
Right	Restituisce un numero specificato di caratteri tratti dalla destra di una stringa.
Rnd	Restituisce un numero casuale compreso tra 0 e 1.
Second	Restituisce la porzione di secondi di un valore orario.
Shell	Esegue un programma.
Space	Restituisce una stringa composta dal numero specificato di spazi.
Split	Divide una stringa in parti, usando un carattere di delimitazione.
Sqr	Restituisce la radice quadrata di un numero.
String	Restituisce un carattere o una stringa ripetuti.
Time	Restituisce l'ora attuale del sistema.
Timer	Restituisce il numero di secondi dalla mezzanotte.
TimeSerial	Restituisce l'ora in base all'ora, al minuto e al secondo specificati.
TimeValue	Converte una stringa in un numero seriale temporale.
Trim	Restituisce una stringa senza spazi iniziali o finali.
TypeName	Restituisce una stringa che descrive il tipo di dati di una variabile.
UBound	Restituisce il più grande indice disponibile per la dimensione di una matrice.
UCase	Converte una stringa in lettere maiuscole.
Val	Restituisce i numeri contenuti in una stringa.
Weekday	Restituisce un numero che rappresenta il giorno della settimana.
Year	Restituisce l'anno da un valore di data.

CONSIGLIO

Per i dettagli completi su una specifica funzione, digitatene il nome in un modulo VBA, portate il cursore su di essa e premete F1.

Usare in VBA le funzioni del foglio di lavoro

Sebbene VBA offra un ampio assortimento di funzioni integrate, potreste non trovare sempre esattamente ciò di cui avete bisogno. Fortunatamente, nelle vostre routine VBA potete anche utilizzare la maggior parte delle funzioni del foglio di lavoro di Excel. Le uniche funzioni che non potete utilizzare sono quelle che hanno una funzione VBA equivalente. Per esempio, non potete utilizzare la funzione RAND (CASUALE) di Excel (che genera un numero casuale) poiché VBA ha una funzione equivalente: Rnd.

VBA rende disponibili le funzioni del foglio di lavoro di Excel tramite l'oggetto WorksheetFunction, contenuto nell'oggetto Application. Ecco un esempio di come potete utilizzare la funzione SUM (SOMMA) di Excel in un'istruzione VBA:

```
Total = Application.WorksheetFunction.SUM(Range("A1:A12"))
```

CONSIGLIO

Potete omettere la parte Application o la parte WorksheetFunction dell'espressione. In entrambi i casi, VBA capisce che cosa state facendo. In altre parole, queste tre espressioni funzionano tutte esattamente allo stesso modo:

```
Total = Application.WorksheetFunction.SUM(Range("A1:A12"))
Total = WorksheetFunction.SUM(Range("A1:A12"))
Total = Application.SUM(Range("A1:A12"))
```

Esempi di funzioni del foglio di lavoro

Molte delle funzioni del foglio di lavoro di Excel possono essere utilizzate nelle espressioni VBA. Potete considerare le funzioni del foglio di lavoro come componenti che potete utilizzare per estendere le vostre routine. In questo paragrafo, scoprirete come incorporare le funzioni del foglio di lavoro di Excel nel vostro codice VBA.

Trovare il valore massimo in un intervallo

Ecco un esempio che mostra come utilizzare la funzione MAX del foglio di lavoro di Excel in una routine VBA. Questa routine visualizza il valore massimo nella colonna A del foglio di lavoro attivo (vedi la Figura 9.3):

```
Sub ShowMax()
     Dim TheMax As Double
     TheMax = WorksheetFunction.MAX(Range("A:A"))
     MsgBox TheMax
End Sub
```

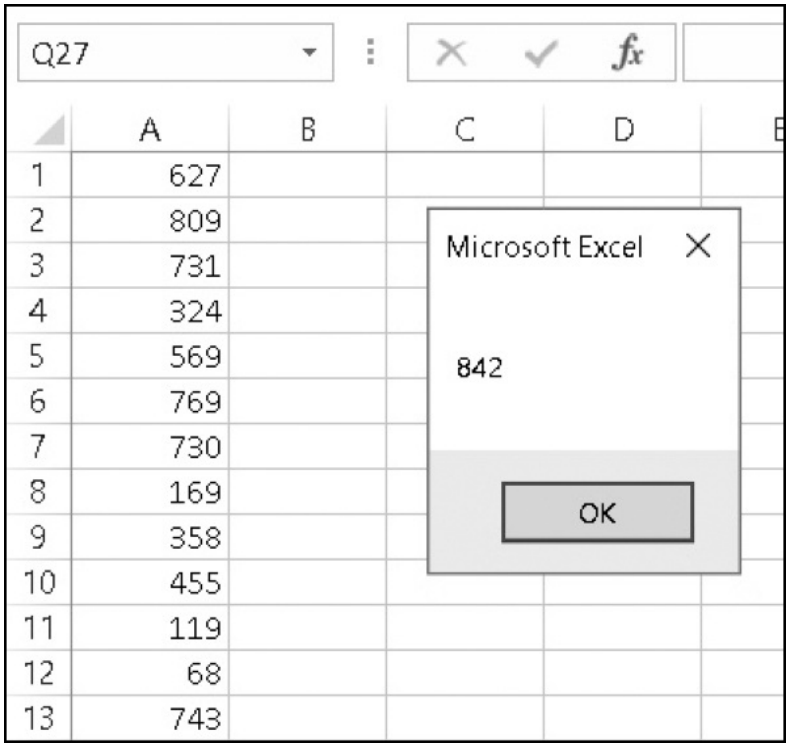

FIGURA 9.3 Uso di una funzione del foglio di lavoro nel codice VBA.

Potete utilizzare la funzione MIN per ottenere il valore più piccolo di un intervallo. E, come potreste aspettarvi, potete utilizzare anche altre funzioni del foglio di lavoro in modo simile. Per esempio, potete utilizzare la funzione LARGE (GRANDE) per determinare il k-*esimo* valore più grande di un intervallo. La seguente espressione lo dimostra:

```
SecondHighest = WorksheetFunction.LARGE(Range("A:A"),2)
```

Notate che la funzione LARGE utilizza due argomenti. Il secondo argomento rappresenta la k-esima parte; in questo caso è 2, quindi il secondo valore più grande.

Calcolare un mutuo

L'esempio seguente utilizza la funzione del foglio di lavoro PMT (RATA) per calcolare la rata di un mutuo. Questa routine utilizza tre variabili per memorizzare i dati passati alla funzione PMT come argomenti. Una finestra MsgBox visualizzerà poi l'importo della rata.

```
Sub PmtCalc()
     Dim IntRate As Double
     Dim LoanAmt As Double
     Dim Periods As Long
     IntRate = 0.0625 / 12
     Periods = 30 * 12
     LoanAmt = 150000
     MsgBox WorksheetFunction.PMT(IntRate, Periods, -LoanAmt)
End Sub
```

Come mostra la seguente istruzione, potete anche inserire i valori direttamente come argomenti della funzione:

```
MsgBox WorksheetFunction.PMT(0.0625 /12, 360, -150000)
```

Tuttavia, l'uso di variabili per memorizzare i parametri rende, se necessario, il codice più facile da leggere e da modificare.

Usare una funzione di ricerca

Nell'esempio seguente vengono utilizzate le funzioni InputBox e MsgBox di VBA, oltre alla funzione VLOOKUP (CERCA.VERT) di Excel. Richiede un numero di parte e poi trae il prezzo da una tabella di ricerca. Nella Figura 9.4, l'intervallo A1:B13 è denominato PriceList.

```
Sub GetPrice()
     Dim PartNum As Variant
     Dim Price As Double
     PartNum = InputBox("Specificare in codice della parte")
     Sheets("Prezzi").Activate
     Price = WorksheetFunction.VLOOKUP(PartNum, Range("Listino"), 2, False)
     MsgBox PartNum & " Prezzo: " & Price
```

```
End Sub
```

FIGURA 9.4 L'intervallo, denominato Listino, contiene i prezzi per le parti.

CONSIGLIO

Potete scaricare la versione originale di questa cartella di lavoro dal sito web del libro.

Ecco come funziona la routine GetPrice:

1. **La funzione InputBox di VBA chiede all'utente un codice di parte.**
2. **Il codice di parte immesso dall'utente viene assegnato alla variabile PartNum.**
3. **L'istruzione successiva attiva il foglio di lavoro Prices (Prezzi), nel caso in cui non sia già attivo.**
4. **Il codice utilizza la funzione VLOOKUP (CERCA.VERT) per trovare il numero di parte nella tabella.**

   Notate che gli argomenti utilizzati in questa istruzione sono quelli che si utilizzerebbero in una formula del foglio di lavoro. Questa istruzione assegna il risultato della funzione alla variabile Price.
5. **Il codice visualizza il prezzo per la parte tramite la funzione MsgBox.**

Questa routine non ha alcuna gestione degli errori, e quindi, se immettete un codice di parte inesistente, fallisce miseramente (provate). Se si trattasse di un'applicazione vera, utilizzata in un'attività reale, dovreste aggiungerle alcune istruzioni in grado di gestire gli errori con maggior grazia. Della gestione degli errori vi parlerò nel Capitolo 12.

Immettere funzioni del foglio di lavoro

Non potete utilizzare la finestra di dialogo Incolla funzione di Excel per copiare una funzione dal foglio di lavoro a un modulo VBA. Dovrete inserite tali funzioni alla "vecchia maniera": a mano. Tuttavia, potete utilizzare la finestra di dialogo Incolla funzione per identificare la funzione che desiderate utilizzare e scoprire i suoi argomenti.

CONSIGLIO

Inoltre potreste usufruire dell'opzione Elenco membri automatico di VBE, che visualizza un elenco a discesa di tutte le funzioni del foglio di lavoro. Basta digitare Application.WorksheetFunction, seguito da un punto. Verrà visualizzato un elenco delle funzioni che potete utilizzare, come mostrato nella Figura 9.5. Se non funziona, selezionate Strumenti ➪ Opzioni, fate clic sulla scheda Editor e attivate un segno di spunta accanto a Elenco membri automatico.

FIGURA 9.5 Ottenere un elenco di funzioni del foglio di lavoro che potete utilizzare nel codice VBA.

Ulteriori informazioni sull'utilizzo delle funzioni del foglio di lavoro

I principianti di VBA spesso confondono le funzioni integrate di VBA con le funzioni della cartella di lavoro di Excel. Una buona regola da ricordare è che VBA non reinventa la ruota. Per la maggior parte, VBA non duplica le funzioni del foglio di lavoro di Excel.

Per molte delle funzioni del foglio di lavoro che non sono disponibili come metodi dell'oggetto WorksheetFunction, potete utilizzare un operatore o una funzione integrata VBA, equivalente. Per esempio, la funzione MOD (RESTO) del foglio di lavoro non è disponibile nell'oggetto WorksheetFunction, perché VBA ha un equivalente: il suo operatore Mod.

Morale? Quando dovete utilizzare una funzione, determinate innanzitutto se VBA ha qualcosa che soddisfi le vostre esigenze. In caso contrario, controllate fra le funzioni del foglio di lavoro. Come ultima istanza, potreste dover scrivere una funzione personalizzata usando il linguaggio VBA.

Utilizzare funzioni personalizzate

Dopo aver visto come gestire le funzioni VBA e le funzioni del foglio di lavoro di Excel, potete entrare nella terza categoria di funzioni utilizzabili nelle routine VBA: le funzioni personalizzate. Le *funzioni personalizzate* (chiamate anche funzioni definite dall'utente) si sviluppano usando (provate a indovinare…) il linguaggio VBA. Per utilizzare una funzione personalizzata, dovete definirla nella cartella di lavoro in cui la utilizzate, oppure dovete definirla in un componente aggiuntivo (consultate il Capitolo 21).

Ecco un esempio di come si definisce una semplice routine Function (MultiplyTwo) e poi la si utilizza in una routine Sub (ShowResult):

```
Function MultiplyTwo(num1, num2) As Double
     MultiplyTwo = num1 * num2
End Function

Sub ShowResult()
     Dim n1 As Double, n2 As Double
     Dim Result As Double
     n1 = 123
     n2 = 544
     Result = MultiplyTwo(n1, n2)
     MsgBox Result
End Sub
```

La funzione personalizzata MultiplyTwo ha due argomenti. La routine Sub ShowResult utilizza questa routine Function passandole due argomenti (tra parentesi). La routine ShowResult visualizza poi una finestra che mostra il valore restituito dalla funzione MultiplyTwo.

Naturalmente, la maggior parte delle funzioni personalizzate non è così banale. Ma questo esempio dovrebbe darvi un'idea di come una routine Sub possa utilizzare una funzione personalizzata.

Le funzioni personalizzate sono utilizzabili anche nelle formule del foglio di lavoro. Per esempio, se avete definito MultiplyTwo nella cartella di lavoro corrente, potrete scrivere una formula come questa:

```
=MultiplyTwo(A1,A2)
```

Questa formula restituisce il prodotto dei valori contenuti nelle celle A1 e A2.

La creazione di funzioni personalizzate è un argomento importante (e molto utile). Così importante che il Capitolo 20 include vari esempi che sono effettivamente utili.

Capitolo 10

Controllare il flusso del programma e prendere decisioni

IN QUESTO CAPITOLO

- **I metodi per controllare il flusso delle routine VBA**
- **La spaventosa istruzione GoTo**
- **Usare le strutture If-Then e Select Case**
- **Esecuzione di cicli nelle routine**

Alcune routine VBA partono dall'inizio del codice e procedono riga per riga fino alla fine, senza mai deviare da questo percorso: dall'alto verso il basso. Le macro che registrate da Excel funzionano sempre così. In molti casi, tuttavia, è necessario controllare il flusso del codice saltando determinate istruzioni, eseguendo più volte un gruppo di istruzioni e verificando una condizione per determinare quale operazione svolgere successivamente. Indossate il casco e godetetevi il viaggio, perché state per scoprire l'essenza della programmazione.

Seguire il flusso

Chi non sa proprio nulla di programmazione non riesce a capire in quale strano modo uno stupido computer possa prendere decisioni intelligenti. Il segreto risiede in vari costrutti di programmazione, supportati dalla maggior parte dei linguaggi di programmazione. La Tabella 10.1 fornisce un'anteprima di questi costrutti (ma poi li spiegherò in dettaglio più avanti in questo capitolo).

TABELLA 10.1 Costrutti di programmazione per prendere decisioni

Costrutto	Che cosa fa
Istruzione GoTo	Salta a un'istruzione.
Struttura If-Then	Fa qualcosa solo se si verifica una determinata condizione.
Seleziona Case	Può fare tutta una serie di cose, a seconda del valore di qualcosa.
Ciclo For-Next	Esegue una sequenza di istruzioni un numero ben preciso di volte.
Ciclo Do-While	Fa qualcosa fintantoché una condizione rimane vera.
Ciclo Do-Until	Fa qualcosa solo fino a quando una condizione diventa vera.

L'istruzione GoTo

Un'istruzione GoTo offre il modo più semplice per modificare il flusso di un programma. L'istruzione GoTo trasferisce semplicemente l'esecuzione del programma a una nuova istruzione, identificata da un'etichetta.

Le routine VBA possono contenere tutte le etichette che volete. Un'etichetta è solo una stringa di testo seguita da due punti.

La seguente routine mostra il funzionamento di un'istruzione GoTo:

```
Sub CheckUser()
     UserName = InputBox("Specificare nome e cognome: ")
     If UserName <> "Maria Rossi" Then GoTo WrongName
     MsgBox ("Benvenuta Maria...")
'   ...[Altro codice vario] ...
     Exit Sub
WrongName:
```

```
      MsgBox "Spiacente. Solo Maria Rossi può proseguire."
End Sub
```

La routine utilizza la funzione InputBox per ottenere il nome dell'utente. Quindi prende una decisione: se l'utente immette un nome diverso da Maria Rossi, il flusso del programma passa all'etichetta WrongName, visualizza un messaggio di scuse e la routine termina. Al contratio, se Maria Rossi esegue questa macro e specifica il proprio nome, la routine visualizza un messaggio di benvenuto ed esegue altro codice aggiuntivo (non mostrato nell'esempio).

Notate che l'istruzione Exit Sub termina la routine prima ancora che la seconda funzione MsgBox abbia la possibilità di funzionare. Senza l'istruzione Exit Sub, verrebbero eseguite entrambe le istruzioni MsgBox.

Questa semplice routine funziona, ma VBA offre diverse alternative migliori (e più strutturate) rispetto a GoTo. In generale, si dovrebbe ricorrere a GoTo solo quando non vi è altro modo di eseguire un'azione. Nella vita reale, l'unica volta che è *davvero necessario* utilizzare un'istruzione GoTo è nella ricerca degli errori (nel Capitolo 12 vedremo una dimostrazione dettagliata della gestione degli errori).

A proposito, la routine CheckUser ha il solo scopo di mostrare il funzionamento dell'istruzione GoTo. Non rappresenta affatto una tecnica di sicurezza efficace!

CHE COS'È LA PROGRAMMAZIONE STRUTTURATA? MI INTERESSA DAVVERO?

Frequentando i programmatori, prima o poi sentirete parlare di *programmazione strutturata*. Questo termine è in uso da decenni e in generale i programmatori concordano sul fatto che i programmi strutturati siano superiori a quelli non strutturati. Quindi che cos'è la programmazione strutturata? E potete usarla in VBA?

La premessa di base della programmazione strutturata è che una routine o un segmento di codice debba avere un solo punto di ingresso e un solo punto di uscita. In altre parole, un blocco di codice dovrebbe essere un'unità indipendente. Un programma non può saltare dentro questa unità; né può uscirne in nessun punto, tranne nel suo singolo punto di uscita. Quando scrivete del codice strutturato, il programma procede in modo ordinato e facile da seguire, a differenza di un programma che salta di qua e di là. Questo, in pratica, esclude l'utilizzo dell'istruzione GoTo.

In generale, un programma strutturato è più facile da leggere e da capire e, cosa più importante, è anche più facile da modificare quando se ne presenterà la necessità.

VBA è un linguaggio strutturato. Offre costrutti strutturati standard, come le istruzioni If-Then-Else, i cicli For-Next, Do-Until e Do-While e le strutture Select Case. Inoltre, supporta pienamente i moduli di codice. Se siete neofiti della programmazione, dovreste provare a sviluppare il più presto possibile le buone abitudini della programmazione strutturata. Ok, fine della lezione.

WARNING

Molti programmatori esperti hanno un odio viscerale per le istruzioni GoTo perché il loro uso tende a risultare difficile da leggere (e difficile da rivedere) producendo una "pastasciutta di codice" (è chiamato proprio *spaghetti code*). Pertanto, non confessate mai di aver utilizzato istruzioni GoTo, parlando con altri programmatori.

Decisioni, sempre decisioni

Come in tanti altri aspetti della vita, anche nella scrittura di macro di Excel, l'efficacia nelle decisioni è la chiave del successo. Questo paragrafo esamina due strutture di programmazione che possono potenziare le vostre routine VBA con alcune capacità decisionali impressionanti: If-Then e Select Case.

La struttura If-Then

If-Then è la struttura di controllo più importante di VBA. Probabilmente userete questo comando più volte tutti i giorni.

Si usa la struttura If-Then quando si desidera eseguire una o più istruzioni in modo condizionale. La clausola opzionale Else, se inclusa, consente di eseguire anche una o più altre istruzioni se la condizione che state controllando *non* è vera. Ecco la semplice routine CheckUser presentata in precedenza in questo capitolo, ricodificata per utilizzare la struttura If-Then-Else:

```
Sub CheckUser2()
      UserName = InputBox("Specificare nome e cognome: ")
      If UserName = "Maria Rossi" Then
          MsgBox ("Benvenuta Maria...")
'       ...[Altro codice vario] ...
```

```
     Else
         MsgBox "Spiacente. Solo Maria Rossi può proseguire."
     End If
End Sub
```

Probabilmente concorderete sul fatto che questa versione è molto più comprensibile.

CONSIGLIO

Dal sito web del libro potete scaricare la cartella di lavoro che contiene tutti gli esempi originali (in inglese) di questo paragrafo.

Esempi d'uso di If-Then

La seguente routine dimostra l'uso della struttura If-Then senza la clausola Else facoltativa:

```
Sub GreetMe()
     If Time < 0.5 Then MsgBox "Buona giornata"
End Sub
```

La routine GreetMe utilizza la funzione VBA Time per ottenere l'ora del sistema. Se l'ora corrente è inferiore a 0,5 (in altre parole, se è prima di mezzogiorno), la routine visualizza un messaggio di benvenuto. Se Time è maggiore o uguale a 0,5, la routine termina e non accade nulla.

Per visualizzare un saluto differente se Time è maggiore o uguale a 0,5, potete aggiungere un'altra istruzione If-Then dopo la prima:

```
Sub GreetMe2()
     If Time < 0.5 Then MsgBox "Buona giornata"
     If Time >= 0.5 Then MsgBox "Buon pomeriggio"
End Sub
```

Notate che la seconda istruzione If-Then utilizza >= (maggiore o uguale a). Questo assicura che sia coperta l'intera giornata. Se avessi usato > (maggiore di), non sarebbe apparso alcun messaggio eseguendo questa routine esattamente a mezzogiorno. È piuttosto improbabile, ma con un programma importante come questo, non vogliamo correre rischi.

Un esempio di If-Then-Else

Un altro approccio al problema precedente utilizza la clausola Else. Ecco la stessa routine ricodificata per utilizzare la struttura If-Then-Else:

```
Sub GreetMe3()
     If Time < 0.5 Then MsgBox "Buona giornata" Else _
         MsgBox "Buon pomeriggio"
End Sub
```

Notate il carattere di continuazione della riga (trattino basso) nell'esempio precedente. L'istruzione If-Then-Else è in realtà una singola istruzione. VBA fornisce anche un modo leggermente diverso per codificare i costrutti If-Then-Else, con un'istruzione finale End If. Pertanto, la routine GreetMe può essere riscritta come:

```
Sub GreetMe4()
     If Time < 0.5 Then
         MsgBox "Buona giornata"
     Else
         MsgBox "Buon pomeriggio"
     End If
End Sub
```

Di fatto, potete inserire un numero qualsiasi di istruzioni sotto la parte If e un numero qualsiasi di istruzioni sotto la parte Else. Questa sintassi è più facile da leggere e rende le istruzioni più corte. Che cosa è necessario fare per espandere la routine GreetMe per gestire tre condizioni: mattina, pomeriggio e sera? Avete due opzioni: usare tre istruzioni If-Then o usare una struttura If-Then-Else *annidata*. *Annidare* significa posizionare una struttura If-Then-Else all'interno di un'altra struttura If-Then-Else. Il primo approccio, utilizzando tre istruzioni If-Then, è più semplice:

```
Sub GreetMe5()
     Dim Msg As String
     If Time < 0.5 Then Msg = "a giornata"
     If Time >= 0.5 And Time < 0.75 Then Msg = " pomeriggio"
     If Time >= 0.75 Then Msg = "asera"
     MsgBox "Buon" & Msg
End Sub
```

Ora potete renderlo elegante usando una variabile. La variabile Msg ottiene un valore di testo diverso, a seconda dell'ora del giorno. L'istruzione MsgBox visualizza il saluto: Buona giornata, Buon pomeriggio o Buonasera.

La seguente routine esegue la stessa azione, ma utilizza una struttura If-Then-End If:

```
Sub GreetMe6()
     Dim Msg As String
     If Time < 0.5 Then
         Msg = "a giornata"
     End If
     If Time >= 0.5 And Time < 0.75 Then
         Msg = " pomeriggio"
     End If
     If Time >= 0.75 Then
         Msg = "asera"
     End If
     MsgBox "Buon" & Msg
End Sub
```

Usare ElseIf

Negli esempi precedenti, viene eseguita ogni istruzione contenuta nella routine, anche al mattino. Una struttura leggermente più efficiente farebbe terminare la routine non appena una condizione risultasse vera. Al mattino, per esempio, la routine dovrebbe visualizzare il messaggio Buona giornata e poi uscire senza valutare le altre due condizioni, superflue.

Con una piccola routine come questa, non è il caso di preoccuparsi della velocità di esecuzione. Ma per applicazioni più grandi, nelle quali la velocità è critica, è necessario conoscere un'altra sintassi per la struttura If-Then: ElseIf.

Ecco come potete riscrivere la routine GreetMe usando questa sintassi:

```
Sub GreetMe7()
     Dim Msg As String
     If Time < 0.5 Then
         Msg = "a giornata"
     ElseIf Time >= 0.5 And Time < 0.75 Then
         Msg = " pomeriggio"
     Else
         Msg = "asera"
     End If
     MsgBox "Buon" & Msg
End Sub
```

Quando una condizione è vera, VBA esegue le istruzioni condizionali e la struttura If termina. In altre parole, VBA non spreca tempo a valutare le condizioni in più, il che rende questa routine un po' più efficiente rispetto agli esempi precedenti. Il compromesso (ci sono sempre compromessi) è che il codice è diventato più difficile da capire (certo, ve ne eravate già accorti).

Un altro esempio If-Then

Ecco un altro esempio che utilizza la forma semplice della struttura If-Then. Questa routine richiede all'utente una quantità e poi visualizza lo sconto appropriato, in base alla quantità immessa dall'utente:

```
Sub ShowDiscount()
     Dim Quantity As Long
     Dim Discount As Double
     Quantity = InputBox("Inserire la quantità:")
     If Quantity > 0 Then Discount = 0.1
     If Quantity >= 25 Then Discount = 0.15
     If Quantity >= 50 Then Discount = 0.2
     If Quantity >= 75 Then Discount = 0.25
     MsgBox "Sconto: " & Discount
End Sub
```

Notate che ciascuna istruzione If-Then di questa routine viene eseguita e il valore di Discount può cambiare quando vengono eseguite le istruzioni. Tuttavia, alla fine la routine visualizza il valore corretto di Discount, perché le istruzioni If-Then sono in ordine di valori di sconto crescente.

La seguente routine esegue le stesse attività utilizzando la sintassi alternativa ElseIf. In questo caso, la routine termina immediatamente dopo l'esecuzione delle istruzioni relative a una condizione vera:

```
Sub ShowDiscount2()
     Dim Quantity As Long
     Dim Discount As Double
     Quantity = InputBox("Inserire la quantità: ")
     If Quantity > 0 And Quantity < 25 Then
         Discount = 0.1
     ElseIf Quantity >= 25 And Quantity < 50 Then
         Discount = 0.15
```

```
    ElseIf Quantity >= 50 And Quantity < 75 Then
        Discount = 0.2
    ElseIf Quantity >= 75 Then
        Discount = 0.25
    End If
    MsgBox "Sconto: " & Discount
End Sub
```

Per quanto le strutture If-Then siano importanti in VBA, diventano ingombranti quando una decisione implica tre o più scelte. Fortunatamente, la struttura Select Case, argomento del prossimo paragrafo, offre un approccio più semplice ed efficiente.

La struttura Select Case

La struttura Select Case è utile per le decisioni che coinvolgono tre o più opzioni (anche se funziona anche con due opzioni, fornendo un'alternativa alla struttura If-Then-Else).

CONSIGLIO

Gli esempi in questo paragrafo sono disponibili (in originale inglese) sul sito web di questo libro.

Un esempio di Select Case

L'esempio seguente mostra come usare la struttura Select Case. Mostra anche un altro modo per codificare gli esempi presentati nel paragrafo precedente:

```
Sub ShowDiscount3()
    Dim Quantity As Long
    Dim Discount As Double
    Quantity = InputBox("Inserire la quantità: ")
    Select Case Quantity
        Case 0 To 24
            Discount = 0.1
        Case 25 To 49
            Discount = 0.15
        Case 50 To 74
            Discount = 0.2
        Case Is >= 75
            Discount = 0.25
    End Select
    MsgBox "Sconto: " & Discount
End Sub
```

In questo esempio, viene valutata la variabile Quantity. La routine verifica quattro diversi casi (Case): 0-24, 25-49, 50-74 e 75 o più.

Dopo ogni istruzione Case potete inserire un numero qualsiasi di istruzioni e, se quel Case è vero, verranno eseguite tutte. Se utilizzate una sola istruzione, come in questo esempio, potete inserire l'istruzione sulla stessa riga della parola riservata Case, preceduta da due punti: il carattere separatore dell'istruzione VBA. Questo tende a rendere il codice più compatto e un po' più chiaro. Ecco come appare la routine in questo formato:

```
Sub ShowDiscount4 ()
    Dim Quantity As Long
    Dim Discount As Double
    Quantity = InputBox("Inserire la quantità: ")
    Select Case Quantity
        Case 0 To 24: Discount = 0.1
        Case 25 To 49: Discount = 0.15
        Case 50 To 74: Discount = 0.2
        Case Is >= 75: Discount = 0.25
    End Select
    MsgBox "Sconto: " & Discount
End Sub
```

Quando VBA esegue una struttura Select Case, la struttura termina non appena VBA trova un Case vero ed esegue le relative istruzioni.

Esempio di Select Case nidificato

Come potete vedere nel seguente esempio, potete nidificare le strutture Select Case. Questa routine esamina la cella attiva e visualizza un messaggio che ne descrive il contenuto. Notate che la routine ha tre strutture Select Case, ciascuna con la propria istruzione End Select:

```
Sub CheckCell()
     Dim Msg As String
       Select Case IsEmpty(ActiveCell)
          Case True
               Msg = "è vuota."
          Case Else
               Select Case ActiveCell.HasFormula
          Case True
               Msg = "contiene una formula."
          Case Else
               Select Case IsNumeric(ActiveCell)
                 Case True
                   Msg = "contiene un numero."
                 Case Else
                   Msg = "contiene un testo."
               End Select
          End Select
       End Select
     MsgBox "La cella " & ActiveCell.Address & " " & Msg
End Sub
```

La logica procede in questo modo:

1. **Scopri se la cella è vuota.**
2. **Se non è vuota, controlla se contiene una formula.**
3. **Se non c'è una formula, scopri se contiene un valore numerico o testuale.**

Al termine della routine, la variabile Msg contiene una stringa che descrive il contenuto della cella. Come illustrato nella Figura 10.1, la funzione MsgBox visualizza quel messaggio.

FIGURA 10.1 Un messaggio visualizzato dalla routine CheckCell.

Potete nidificare le strutture Select Case nel modo necessario, ma assicuratevi che a ogni istruzione Select Case corrisponda un'istruzione End Select.

CONSIGLIO

Se non siete ancora convinti che valga la pena di usare l'indentazione del codice, il listato precedente funge da buon esempio. L'identazione rende davvero più chiari i livelli di nidificazione. Date un'occhiata alla stessa identica routine, ma scritta senza indentazione:

```
Sub CheckCell()
Dim Msg As String
Select Case IsEmpty(ActiveCell)
Case True
```

```
Msg = "è vuota."
Case Else
Select Case ActiveCell.HasFormula
Case True
Msg = "contiene una formula."
Case Else
Select Case IsNumeric(ActiveCell)
Case True
Msg = "contiene un numero."
Case Else
Msg = "contiene un testo."
End Select
End Select
End Select
MsgBox "Cell " & ActiveCell.Address & " " & Msg
End Sub
```

Abbastanza incomprensibile, vero?

I cicli nel codice

Il termine *ciclo* fa riferimento alla ripetizione di un blocco di istruzioni VBA più e più volte. Perché si dovrebbero usare i cicli? Il vostro codice può eseguire un ciclo…

- su un intero intervallo di celle, lavorando su ogni singola cella;
- su tutte le cartelle di lavoro aperte (la collezione Workbooks) e su ognuna eseguire qualcosa;
- su tutti i fogli di lavoro di una cartella di lavoro (la collezione Worksheets) e su ognuno di essi eseguire qualcosa;
- su tutti gli elementi di una matrice (un array);
- su tutti i caratteri di una cella;
- su tutti i grafici di un foglio di lavoro (la collezione ChartObjects) e su ognuno di essi eseguire qualcosa.

Il linguaggio VBA supporta diversi tipi di ciclo e gli esempi che seguono illustrano alcuni modi in cui potete utilizzarli.

I cicli For-Next

CONSIGLIO

Gli esempi in questo paragrafo sono disponibili (in originale inglese) sul sito web di questo libro.

Il tipo più semplice di ciclo è For-Next. Il ciclo è controllato da una variabile contatore, che inizia con un valore e termina con un altro. Le istruzioni tra l'istru zione For e l'istruzione Next sono quelle che vengono ripetute nel ciclo. Per capirne il funzionamento, continuate a leggere.

Un esempio di ciclo For-Next

Nell'esempio seguente viene utilizzato un ciclo For-Next per sommare i primi 1.000 numeri positivi. La variabile Total inizia con il valore zero (0). Quindi si apre il ciclo. La variabile Cnt è il contatore del ciclo. Inizia da 1 e viene incrementata di un'unità ogni volta che viene eseguito il ciclo. Il ciclo termina quando Cnt è uguale a 1.000.

Questo esempio ha solo un'istruzione all'interno del ciclo. Questa istruzione aggiunge il valore di Cnt alla variabile Total. Al termine del ciclo, una MsgBox visualizza la somma dei numeri.

```
Sub AddNumbers ()
     Dim Total As Double
     Dim Cnt As Long
     Total = 0
     For Cnt = 1 To 1000
         Total = Total + Cnt
     Next Cnt
     MsgBox Total
End Sub
```

WARNING

Poiché il contatore del ciclo è una comune variabile, potete scrivere del codice per modificarne il valore all'interno del blocco di codice tra le istruzioni For e Next. Questa, tuttavia, è una *pessima* idea. La modifica del contatore all'interno del ciclo può dare risultati imprevedibili. Adottate precauzioni speciali per garantire che il vostro codice non modifichi il valore del contatore del ciclo.

Esempi di cicli For-Next con Step

Potete utilizzare un valore Step per saltare alcuni valori del contatore di un ciclo For-Next. Ecco l'esempio precedente, riscritto per sommare solo i numeri dispari compresi tra 1 e 1.000:

```
Sub AddOddNumbers()
     Dim Total As Double
     Dim Cnt As Long
     Total = 0
     For Cnt = 1 To 1000 Step 2
         Total = Total + Cnt
     Next Cnt
     MsgBox Total
End Sub
```

Questa volta, Cnt inizia da 1 e poi assume i valori 3, 5, 7 e così via. Il valore Step determina il modo in cui viene incrementato il contatore. Notate che il valore del ciclo superiore (1000) in effetti non viene utilizzato, perché il massimo valore di Cnt sarà 999.

Ecco un altro esempio, che utilizza un valore Step pari a 3. Questa routine funziona sul foglio attivo e applica un'ombreggiatura di colore grigio chiaro a ogni terza riga, dalla riga 1 alla riga 100.

```
Sub ShadeEveryThirdRow()
     Dim i As Long
     For i = 1 To 100 Step 3
         Rows(i).Interior.Color = RGB(200, 200, 200)
     Next i
End Sub
```

La Figura 10.2 mostra il risultato dell'esecuzione di questa macro.

FIGURA 10.2 Utilizzo di un ciclo per applicare l'ombreggiatura di sfondo alle righe.

Un esempio di ciclo For-Next con Exit For

Un ciclo For-Next può anche includere, all'interno del ciclo, una o più istruzioni Exit For. Quando VBA rileva questa istruzione, il ciclo termina immediatamente.

L'esempio seguente, disponibile sul sito web del libro, illustra l'istruzione Exit For. Questa è una routine Function, progettata per essere utilizzata in una formula del foglio di lavoro. La funzione accetta un argomento (una variabile denominata Str) e restituisce i caratteri che si trovano a sinistra della prima cifra numerica. Per esempio, se

l'argomento è "KBR98Z", la funzione restituisce "KBR".

```
Function TextPart(Str)
     Dim i As Long
     TextPart = ""
     For i = 1 To Len(Str)
         If IsNumeric(Mid(Str, i, 1)) Then
              Exit For
         Else
              TextPart = TextPart & Mid(Str, i, 1)
         End If
     Next i
End Function
```

Il ciclo For-Next inizia con 1 e termina con il numero che rappresenta il numero di caratteri nella stringa. Il codice utilizza la funzione Mid di VBA per estrarre un singolo carattere all'interno del ciclo. Se viene trovato un carattere numerico, viene eseguita l'istruzione Exit For e il ciclo termina prematuramente. Se il carattere non è numerico, viene aggiunto al valore restituito (che ha lo stesso nome della funzione). L'unico caso in cui il ciclo esaminerà ogni carattere si verifica quando la stringa passata come argomento non contiene caratteri numerici.

Ora, potreste dire: "Ehi, ma prima avevi detto qualcosa sul fatto di usare sempre un solo punto di uscita!". Avete ragione, e ovviamente, questo vuol dire che state imparando l'arte della programmazione strutturata. Ma in alcuni casi, ignorare questa regola è una decisione saggia. In questo esempio, si ottiene una notevole accelerazione del codice, perché non c'è motivo di far proseguire il ciclo dopo aver trovato la prima cifra numerica.

Un esempio di ciclo For-Next annidato

Finora, tutti gli esempi del capitolo hanno utilizzato cicli relativamente semplici. Tuttavia, potete avere nel ciclo un numero qualsiasi di istruzioni e annidare dei cicli For-Next all'interno di altri cicli For-Next.

L'esempio seguente utilizza un ciclo For-Next annidato per inserire numeri casuali in un intervallo di celle di 12 righe per 5 colonne, come mostrato nella Figura 10.3. Notate che la routine esegue il *ciclo interno* (il ciclo con il contatore Row) una volta per ogni iterazione del *ciclo esterno* (il ciclo con il contatore Col). In altre parole, la routine esegue l'istruzione Cells (Row, Col) = Rnd per sessanta volte.

```
Sub FillRange()
     Dim Col As Long
     Dim Row As Long
     For Col = 1 To 5
         For Row = 1 To 12
              Cells(Row, Col) = Rnd
         Next Row
     Next Col
End Sub
```

L'esempio seguente utilizza i cicli For-Next nidificati per inizializzare una matrice tridimensionale con il valore 100. Questa routine esegue l'istruzione annidata dentro tutti i cicli (l'istruzione di assegnamento) per 1.000 volte (10 * 10 * 10), ogni volta con una diversa combinazione di valori per i, j, e k:

```
Sub NestedLoops()
     Dim MyArray(10, 10, 10)
     Dim i As Long
     Dim j As Long
     Dim k As Long
     For i = 1 To 10
         For j = 1 To 10
              For k = 1 To 10
                MyArray(i, j, k) = 100
              Next k
         Next j
     Next i
'   Altre istruzioni
End Sub
```

Per ulteriori informazioni sulle matrici fate riferimento al Capitolo 7.

	A	B	C	D	E	F
1	0,11547	0,746007	0,734488	0,959271	0,557501	
2	0,173773	0,08856	0,277168	0,542029	0,909204	
3	0,048119	0,634571	0,356586	0,494377	0,657256	
4	0,714816	0,713028	0,433457	0,972967	0,441115	
5	0,533022	0,015682	0,944971	0,2178	0,692995	
6	0,561009	0,431144	0,12155	0,378991	0,064491	
7	0,216734	0,401983	0,645951	0,395844	0,756093	
8	0,468006	0,275295	0,347833	0,281503	0,700514	
9	0,746355	0,985374	0,104432	0,503371	0,497071	
10	0,752311	0,802587	0,185403	0,138732	0,155436	
11	0,398932	0,696101	0,077685	0,517256	0,223738	
12	0,903099	0,417599	0,43276	0,965364	0,326144	

FIGURA 10.3 Queste celle sono state compilate utilizzando un ciclo For-Next annidato.

Per informazioni sulle matrici, consultate il Capitolo 7.

Ecco un esempio finale che utilizza dei cicli For-Next annidati con un valore Step. Questa routine crea una scacchiera, cambiando il colore di sfondo delle celle alternate (vedi la Figura 10.4).

Il contatore Row esegue i cicli da 1 a 8. Un costrutto If-Then determina quale struttura For-Next annidata usare. Per le righe dispari, il contatore Col parte da 2. Per le righe pari, il contatore Col parte da 1. Entrambi i cicli utilizzano un valore Step di 2, quindi vengono sempre interessate delle celle alternate. Due ulteriori istruzioni rendono le celle quadrate (proprio come una vera scacchiera).

```
Sub MakeCheckerboard()
     Dim R As Long, C As Long
     For R = 1 To 8
         If WorksheetFunction.IsOdd(R) Then
              For C = 2 To 8 Step 2
                Cells(R, C).Interior.Color = 255
              Next C
         Else
              For C = 1 To 8 Step 2
                Cells(R, C).Interior.Color = 255
              Next C
         End If
     Next R
     Rows("1:8").RowHeight = 35
     Columns("A:H").ColumnWidth = 6.5
End Sub
```

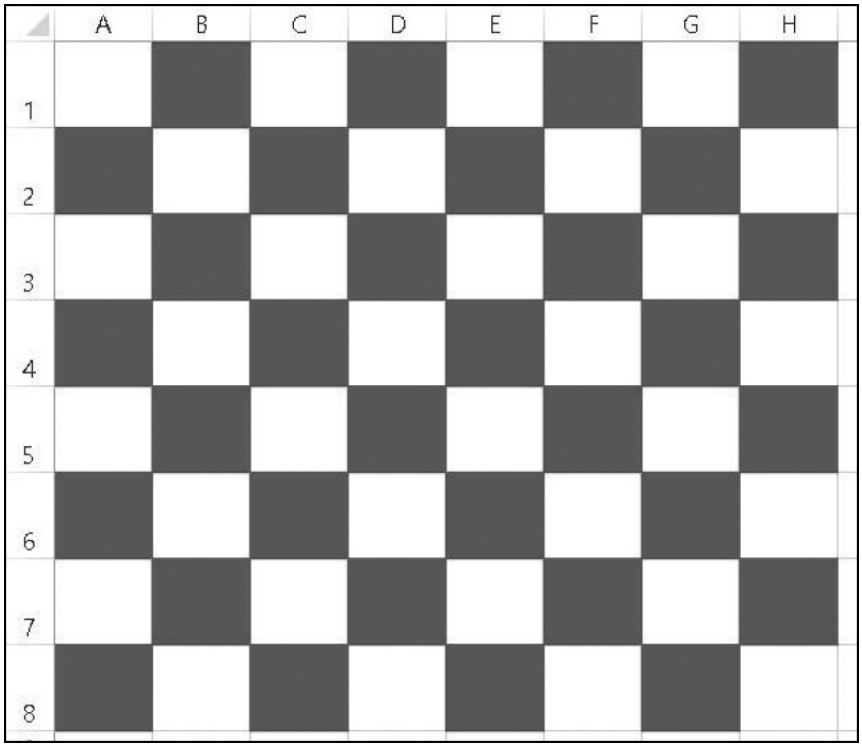

FIGURA 10.4 Uso dei cicli per creare una scacchiera.

Il ciclo Do-While

Il linguaggio VBA supporta un altro tipo di struttura ciclica, nota come ciclo Do-While. A differenza di un ciclo For-Next, un ciclo Do-While viene eseguito fintantoché è soddisfatta una determinata condizione.

Nell'esempio seguente viene utilizzato un ciclo Do-While. Questa routine utilizza la cella attiva come punto di partenza e poi scorre lungo la colonna, moltiplicando il valore di ciascuna cella per 2. Il ciclo continua finché la routine trova celle piene.

```
Sub DoWhileDemo()
     Do While ActiveCell.Value <> Empty
         ActiveCell.Value = ActiveCell.Value * 2
         ActiveCell.Offset(1, 0).Select
     Loop
End Sub
```

Il ciclo Do-Until

Un ciclo Do-Until è simile a un ciclo Do-While. Le due strutture si differenziano per la gestione della condizione di test. Un programma continua a eseguire un ciclo Do-While *finché* la condizione rimane vera. In un ciclo Do-Until, il programma esegue il ciclo solo *fino a* quando la condizione diventa vera.

L'esempio seguente è uguale a quello per il ciclo Do-While, ma ricodificato per utilizzare un ciclo Do-Until:

```
Sub DoUntilDemo()
     Do Until IsEmpty(ActiveCell.Value)
         ActiveCell.Value = ActiveCell.Value * 2
         ActiveCell.Offset(1, 0).Select
     Loop
End Sub
```

Usare i cicli For Each-Next con le collezioni

Il linguaggio VBA supporta ancora un altro tipo di ciclo: il ciclo su ogni oggetto di una collezione. Una collezione, come forse saprete, consiste in un numero di oggetti dello stesso tipo. Per esempio, Excel ha una collezione di tutte le cartelle di lavoro aperte (la collezione Workbooks) e ogni cartella di lavoro ha una collezione di fogli di lavoro (la collezione Worksheets).

CONSIGLIO

Gli esempi in questo paragrafo sono disponibili (in originale inglese) sul sito web di questo libro.

Quando, in una collezione, è necessario eseguire un'operazione su ogni oggetto, si usa la struttura For Each-Next. L'esempio seguente analizza ogni foglio di lavoro della cartella di lavoro attiva e, se è vuoto, lo cancella:

```
Sub DeleteEmptySheets()
     Dim WkSht As Worksheet
     Application.DisplayAlerts = False
     For Each WkSht In ActiveWorkbook.Worksheets
         If WorksheetFunction.CountA(WkSht.Cells) = 0 Then
              WkSht.Delete
         End If
     Next WkSht
     Application.DisplayAlerts = True
End Sub
```

In questo esempio, WkSht è una variabile oggetto che rappresenta ciascun foglio di lavoro nella cartella di lavoro. La scelta del nome WkSht è del tutto casuale; potete usare il nome che preferite.

Il codice esamina ogni foglio di lavoro e determina se è vuoto contando le celle non vuote. Se il conteggio è pari a zero, il foglio è decisamente vuoto e viene eliminato. Notate che, mentre il ciclo è in azione, l'impostazione DisplayAlerts è disattivata. Senza questa istruzione, Excel visualizzerebbe un avviso ogni volta che un foglio sta per essere cancellato.

Se tutti i fogli di lavoro nella cartella di lavoro sono vuoti, quando Excel tenterà di eliminare l'ultimo (e unico) foglio, si verificherà un errore. Normalmente, si dovrebbe scrivere del codice per gestire tale situazione.

Ecco un altro esempio di For Each-Next. Questa routine utilizza un ciclo per nascondere tutti i fogli di lavoro della cartella di lavoro attiva, tranne il foglio attivo.

```
Sub HideSheets()
     Dim Sht As Worksheet
     For Each Sht In ActiveWorkbook.Worksheets
         If Sht.Name <> ActiveSheet.Name Then
              Sht.Visible = xlSheetHidden
         End If
     Next Sht
End Sub
```

La routine HideSheets controlla il nome del foglio. Se è diverso dal nome del foglio attivo, lo nasconde. Notate che la proprietà Visible non è booleana. In effetti questa proprietà può assumere *tre* valori ed Excel fornisce tre costanti predefinite. Se siete curiosi di conoscere la terza possibilità (xlVeryHidden), controllate la Guida.

Ciò che viene nascosto deve poter essere scoperto, quindi ecco una macro che mostra tutti i fogli di lavoro presenti nella cartella di lavoro attiva:

```
Sub UnhideSheets()
     Dim Sht As Worksheet
     For Each Sht In ActiveWorkbook.Worksheets
         Sht.Visible = xlSheetVisible
     Next Sht
End Sub
```

Come immaginate, potete creare anche cicli For Each-Next annidati. La routine CountBold esamina ogni cella dell'intervallo in uso di ciascun foglio di lavoro di ogni cartella di lavoro aperta e visualizza il numero di celle formattate in grassetto:

```
Sub CountBold()
     Dim WBook As Workbook
     Dim WSheet As Worksheet
     Dim Cell As Range
     Dim Cnt As Long
     For Each WBook In Workbooks
         For Each WSheet In WBook.Worksheets
              For Each Cell In WSheet.UsedRange
                If Cell.Font.Bold = True Then Cnt = Cnt + 1
              Next Cell
         Next WSheet
     Next WBook
     MsgBox Cnt & " celle in grassetto."
End Sub
```

Capitolo 11

Routine ed eventi automatici

IN QUESTO CAPITOLO

- **I tipi di eventi che possono attivare un'esecuzione**
- **Dove posizionare il codice del gestore di eventi VBA**
- **Eseguire una macro all'apertura o alla chiusura di una cartella di lavoro**
- **Eseguire una macro quando si richiama una cartella o un foglio di lavoro**

Potete eseguire una routine Sub in vari modi. Un modo consiste nel far sì che la routine Sub venga eseguita automaticamente. In questo capitolo, esploreremo questa funzionalità potenzialmente utile, spiegando come impostare le cose in modo che una macro venga eseguita automaticamente al verificarsi di un determinato evento.

Preparativi per il grande evento

Un *evento* è, fondamentalmente, qualcosa che succede in Excel. Di seguito sono riportati alcuni esempi dei tipi di eventi che Excel può riconoscere:

- viene aperta o chiusa una cartella di lavoro;
- viene attivata o disattivata una finestra;
- viene attivato o disattivato un foglio di lavoro;
- in una cella vengono inseriti dei dati o comunque la cella viene modificata;
- viene salvata una cartella di lavoro;
- viene fatto clic su un oggetto, per esempio un pulsante;
- viene premuto un particolare tasto o una combinazione di tasti;
- viene raggiunta una determinata ora del giorno;
- si verifica un errore.

A ogni genere di oggetti di Excel (cartelle di lavoro, fogli di lavoro, tabelle pivot e perfino grafici) vengono associati decine di diversi tipi di eventi. Per semplificare le cose, potete utilizzare altri due tipi più comuni di eventi: gli eventi della cartella di lavoro e gli eventi del foglio di lavoro.

La Tabella 11.1 elenca alcuni eventi utili relativi alla cartella di lavoro. Se, per qualche motivo, avete bisogno di conoscere l'elenco completo degli eventi relativi alla cartella di lavoro, controllate la Guida.

TABELLA 11.1 Eventi della cartella di lavoro

Evento	Quando viene attivato
Activate	La cartella di lavoro viene attivata.
BeforeClose	Viene eseguito il comando per chiudere la cartella di lavoro.
BeforePrint	Viene eseguito il comando di stampa.
BeforeSave	Viene eseguito il comando di salvataggio della cartella di lavoro.
Deactivate	La cartella di lavoro è stata disattivata.
NewSheet	Alla cartella di lavoro viene aggiunto un nuovo foglio.
Open	La cartella di lavoro viene aperta.
SheetActivate	Nella cartella di lavoro viene attivato un foglio.
SheetBeforeDoubleClick	Viene fatto doppio clic su una cella nella cartella di lavoro.

SheetBeforeRightClick	Viene fatto clic destro su una cella della cartella di lavoro.
SheetChange	A una cella della cartella di lavoro viene apportata una modifica.
SheetDeactivate	Nella cartella di lavoro viene disattivato un foglio.
SheetSelectionChange	La selezione è cambiata.
WindowActivate	La finestra della cartella di lavoro viene attivata.
WindowDeactivate	La finestra della cartella di lavoro viene disattivata.

La Tabella 11.2 elenca alcuni eventi utili relativi al foglio di lavoro.

TABELLA 11.2 Eventi del foglio di lavoro.

Evento	Quando viene attivato
Activate	Il foglio di lavoro viene attivato.
BeforeDoubleClick	Su una cella del foglio di lavoro viene fatto doppio clic.
BeforeRightClick	Su una cella del foglio di lavoro viene fatto clic destro.
Change	A una cella del foglio di lavoro viene apportata una modifica.
Deactivate	Il foglio di lavoro viene disattivato.
SelectionChange	La selezione è cambiata.

Gli eventi sono utili?

A questo punto, vi starete chiedendo in che modo questi eventi possono essere utili. Ecco un rapido esempio.

Immaginate di avere una cartella di lavoro in cui inserite dei dati nella colonna A. Il vostro capo, una persona molto nervosa, vi dice che ha bisogno di sapere esattamente quando è stato inserito ciascun valore. L'inserimento dei dati è un evento: un evento WorksheetChange. Potete scrivere una macro che risponda a questo evento. Quella macro si attiva ogni volta che il foglio di lavoro viene modificato. Se la modifica è stata apportata nella colonna A, la macro inserisce la data e l'ora nella colonna B, accanto al dato inserito.

Forse, a questo punto, sarete curiosi di sapere che aspetto avrebbe tale macro. Scommetto che è molto più semplice di quanto immaginavate:

```
Private Sub Worksheet_Change(ByVal Target As Range)
     If Target.Column = 1 Then
         Target.Offset(0, 1) = Now
     End If
End Sub
```

Tra l'altro, le macro che rispondono agli eventi sono molto esigenti riguardo alla posizione in cui devono essere memorizzate. Per esempio, la macro Worksheet_Change *deve* essere nel modulo di codice di quel foglio di lavoro. Collocandola da qualche altra parte non funzionerà. Più avanti, sempre in questo capitolo, vi daremo ulteriori informazioni (vedere il paragrafo "Dove va messo il codice VBA?").

WARNING

Il solo fatto che la vostra cartella di lavoro contiene routine che rispondono agli eventi non garantisce che tali routine vengano effettivamente eseguite. Come sapete, potete aprire una cartella di lavoro chiedendo di disabilitare le macro. In tal caso, tutte le macro (comprese le routine che rispondono agli eventi) saranno disattivate. Tenetelo presente quando create cartelle di lavoro che fanno affidamento su routine gestite da eventi.

Programmare routine di gestione degli eventi

Una routine VBA che viene eseguita in risposta a un evento è denominata *routine di gestione degli eventi* o *gestori di eventi*. Si tratta sempre di routine Sub (quindi non Function). Scrivere questi gestori di eventi è relativamente semplice, dopo aver compreso come funziona l'operazione.

La creazione di routine di gestione degli eventi si riduce a pochi passaggi.

1. **Identificate l'evento per il quale desiderate attivare la routine.**
2. **Premete Alt + F11 per attivare Visual Basic Editor.**
3. **Nella finestra Progetto di VBE, fate doppio clic sull'oggetto appropriato elencato in Microsoft Excel Oggetti.**

Per gli eventi relativi alla cartella di lavoro, l'oggetto è Questa_cartella_di_lavoro. Per un evento correlato al foglio di lavoro, l'oggetto è un oggetto Worksheet (come Foglio1).

4. **Nella finestra Codice dell'oggetto, scrivete la routine di gestione degli eventi che deve essere eseguita quando si verifica l'evento.**

   Questa routine avrà un nome speciale che la identifica come una routine di gestione degli eventi.

Questi passaggi diventeranno più chiari proseguendo la lettura del capitolo.

Dove va messo il codice VBA?

È molto importante capire dove devono trovarsi le routine del vostro nuovo gestore di eventi. Devono risiedere nella finestra Codice di un modulo di Microsoft Excel Oggetti. Non vanno inserite in un modulo VBA standard. Se inserite la routine del vostro gestore di eventi nel posto sbagliato, semplicemente non funzionerà. E non verrà visualizzato neanche un messaggio d'errore.

La Figura 11.1 mostra la finestra di VBE con un progetto visualizzato nel riquadro Progetto (per informazioni sul VBE fate riferimento al Capitolo 3). Notate che il progetto VBA per Cartel1 è completamente espanso e comprende diversi oggetti:

- un oggetto per ogni foglio di lavoro contenuto nella cartella di lavoro (in questo caso, tre oggetti Foglio).
- un oggetto con etichetta Questa_cartella_di_lavoro.
- un modulo VBA inserito manualmente (selezionando Inserisci ➪ Modulo).

FIGURA 11.1 La finestra Progetto visualizza gli elementi per un singolo progetto.

Facendo doppio clic su uno qualsiasi di questi oggetti, viene visualizzato il codice associato all'elemento, se presente.

Le routine dei gestori di eventi che scrivete vanno nella finestra Codice dell'elemento Questa_cartella_di_lavoro (per gli eventi correlati alla cartella di lavoro) o di uno degli oggetti Foglio (per gli eventi relativi ai fogli di lavoro).

Nella Figura 11.1, viene visualizzata la finestra Codice per l'oggetto Foglio1, e si dà il caso che sia definita una singola routine di gestione degli eventi. Notate i due controlli a discesa nella parte superiore del modulo Codice? Quelli con scritto Worksheet e Change? Continuate a leggere per scoprire quanto sono utili.

Scrivere una routine di gestione degli eventi

Il VBE vi viene in aiuto nella scrittura di una routine di gestione degli eventi: visualizza un elenco di tutti gli eventi relativi all'oggetto selezionato.
Nella parte superiore di ogni finestra del codice, trovate due elenchi a discesa:

- l'elenco a discesa Oggetto (a sinistra);
- l'elenco a discesa Routine (a destra).

Come impostazione predefinita, l'elenco a discesa Oggetto nella finestra Codice visualizza l'indicazione Generale.

Se state scrivendo un gestore di eventi per l'oggetto Questa_cartella_di_lavoro, dovete fate doppio clic su Questa_cartella_di_lavoro nel riquadro Progetto e poi scegliere Workbook dal menu a discesa Oggetto (è l'unica altra scelta).

Se state scrivendo un gestore di eventi per un oggetto Foglio, dovete fare doppio clic sul foglio nel riquadro Progetto e poi scegliere Worksheet dall'elenco a discesa Oggetto (di nuovo, l'unica altra scelta).

Dopo aver effettuato la scelta dall'elenco a discesa Oggetto, potete scegliere l'evento dall'elenco a discesa Procedure. La Figura 11.2 mostra alcune delle scelte disponibili per un evento correlato alla cartella di lavoro.

FIGURA 11.2 Scelta di un evento nella finestra Codice per l'oggetto Questa_cartella_di_lavoro.

Quando selezionate un evento dall'elenco, VBE inizia automaticamente a creare una routine di gestione degli eventi. Questa è una funzionalità molto utile, perché vi dice esattamente quali sono gli argomenti corretti.

Ecco un piccolo cavillo. Quando selezionate per la prima volta la cartella di lavoro dall'elenco degli oggetti, il VBE presuppone sempre che desideriate creare una routine di gestione per l'evento Open e la crea per voi. Se state effettivamente creando una routine Workbook_Open, va bene. Tuttavia, se state creando una routine per un altro evento, dovrete eliminare la routine Workbook_Open vuota creata.

VBE, tuttavia, si ferma qui. Scrive l'istruzione Sub e l'istruzione End Sub. Scrivere il codice VBA che andrà tra queste due istruzioni è compito vostro.

RICORDA!

Non è necessario utilizzare questi due elenchi a discesa, ma è anche vero che la loro presenza vi semplifica la vita, perché il nome della routine di gestione degli eventi è di fondamentale importanza. Se il nome è sbagliato, la routine non funzionerà. Inoltre, alcune routine di gestione degli eventi utilizzano, nell'istruzione Sub, uno o più argomenti. Non c'è modo di ricordare quali sono questi argomenti. Per esempio, se selezionate SheetActivate dall'elenco eventi per un oggetto Workbook, VBE scrive la seguente istruzione Sub:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
```

In questo caso, Sh è l'argomento passato alla routine ed è una variabile che rappresenta il foglio nella cartella di lavoro attiva. Gli esempi in questo capitolo chiariscono questo punto.

Esempi introduttivi

In questo paragrafo, troverete alcuni esempi utili per comprendere meglio questo meccanismo.

L'evento Open per una cartella di lavoro

Uno degli eventi più comunemente utilizzati è l'evento Workbook Open: l'apertura di una cartella di lavoro. Supponiamo che abbiate una cartella di lavoro che utilizzate ogni giorno. In questo esempio, la routine Workbook_Open viene eseguita ogni volta che la cartella di lavoro viene aperta. La routine controlla il giorno della settimana; se è venerdì, il codice visualizza un promemoria.

Ecco come creare una routine che viene eseguita ogni volta che si verifica l'evento Work book Open.

1. **Aprite la cartella di lavoro.**

   Andrà bene una cartella di lavoro qualsiasi.

2. **Premete Alt + F11 per attivare il VBE.**
3. **Individuate la cartella di lavoro nel riquadro Progetto.**
4. **Se necessario, fate doppio clic sul nome del progetto, per visualizzarne gli elementi.**
5. **Fate doppio clic sull'elemento Questa_cartella_di_lavoro.**

   Il VBE aprirà una finestra di codice vuota per l'oggetto Questa_cartella_di_lavoro.

6. **Nella finestra Codice, selezionate Workbook dall'elenco a discesa Oggetto (a sinistra).**

VBE immette le istruzioni di inizio e fine routine per Workbook_Open.

7. **Digitate le istruzioni della routine, in modo che la routine, finita, somigli alla seguente:**

```
Private Sub Workbook_Open()
     Dim Msg As String
     If Weekday(Now) = 6 Then
         Msg = "Oggi è venerdì. Non dimenticarti di "
         Msg = Msg & "preparare il report TPS!"
         MsgBox Msg
     End If
End Sub
```

La finestra del codice dovrebbe somigliare a quanto è rappresentato nella Figura 11.3.

FIGURA 11.3 La routine del gestore di questo evento viene eseguita all'apertura della cartella di lavoro.

La routine Workbook_Open viene eseguita automaticamente a ogni apertura della cartella di lavoro. La funzione WeekDay di VBA determina il giorno della settimana. Se è venerdì (il giorno 6), un messaggio ricorda all'utente di inviare un rapporto. Se non è venerdì, non succede nulla.

Se oggi non è venerdì, potreste avere difficoltà a provare questa routine. Per farlo, potete semplicemente cambiare il valore 6 con quello corrispondente al numero del giorno odierno.

E, naturalmente, potete modificare questa routine in qualsiasi modo desideriate. La seguente versione, per esempio, visualizza un messaggio a ogni apertura della cartella di lavoro. Alla lunga, la cosa diventa fastidiosa.

```
Private Sub Workbook_Open()
     Msg = "Benvenuti alla cartella di lavoro di Frank!"
     MsgBox Msg
End Sub
```

Una routine Workbook_Open può fare quasi tutto. Questi gestori di eventi vengono spesso utilizzati per:

- visualizzare messaggi di benvenuto (come nel listato precedente);
- aprire altre cartelle di lavoro;
- attivare un determinato foglio di lavoro nella cartella di lavoro;
- impostare dei menu contestuali personalizzati.

Ecco un ultimo esempio di routine Workbook_Open che utilizza le funzioni GetSetting e SaveSetting per tenere traccia di quante volte è stata aperta la cartella di lavoro. La funzione SaveSetting scrive un valore nel registro di Windows e la funzione GetSetting recupera tale valore (per i dettagli consultate la Guida).

L'esempio seguente recupera il conteggio dal registro, lo incrementa e poi lo salva di nuovo nel registro. Indica inoltre all'utente il valore di Cnt corrispondente al numero di volte in cui la cartella di lavoro è stata aperta (vedi la Figura 11.4).

```
Private Sub Workbook_Open()
     Dim Cnt As Long
     Cnt = GetSetting("MyApp", "Settings", "Open", 0)
     Cnt = Cnt + 1
     SaveSetting "MyApp", "Settings", "Open", Cnt
     MsgBox "Questa cartella è stata aperta " & Cnt & " volte."
End Sub
```

FIGURA 11.4 Utilizzo di un gestore di eventi Workbook_Open per tenere traccia di quante volte è stata aperta una cartella di lavoro.

L'evento BeforeClose per una cartella di lavoro

Ecco un esempio della routine di gestione degli eventi Workbook_BeforeClose, che viene eseguita automaticamente immediatamente prima che la cartella di lavoro venga chiusa. Questa routine si trova nella finestra Codice per un oggetto Questa_cartella_di_lavoro:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
     Dim Msg As String
     Dim Ans As Long
     Dim FName As String
     Msg = "Intendi creare una copia di backup di questo file?"
     Ans = MsgBox(Msg, vbYesNo)
     If Ans = vbYes Then
         FName = "F:\BACKUP\" & Questa_cartella_di_lavoro.Name
         Questa_cartella_di_lavoro.SaveCopyAs FName
     End If
End Sub
```

Questa routine utilizza una finestra per chiedere all'utente se desidera eseguire una copia di backup della cartella di lavoro. Se la risposta è sì, il codice utilizza il metodo SaveCopyAs per salvare una copia di backup del file nell'unità F. Per adattare questa routine, dovrete quindi modificare l'unità dischi e il percorso.

I programmatori Excel usano spesso una routine Workbook_BeforeClose per mettere in ordine le cose. Per esempio, se utilizzate una routine Workbook_Open per modificare alcune impostazioni all'apertura di una cartella di lavoro (nascondendo, per esempio, la barra di stato), è solo ovvio che, alla chiusura della cartella di lavoro, dobbiate ripristinare le impostazioni allo stato originale. Potete eseguire questa pulizia elettronica con una routine per Workbook_BeforeClose.

WARNING

Quando utilizzate l'evento Workbook_BeforeClose, tenete presente quanto segue: se chiudete Excel e, dall'ultimo salvataggio, è stato modificato un file aperto, Excel mostra il normale messaggio "Salva le modifiche". Facendo clic sul pulsante Annulla, annullerete l'intero processo di chiusura. Ma l'evento Workbook_BeforeClose verrà comunque eseguito.

L'evento BeforeSave per una cartella di lavoro

L'evento BeforeSave si verifica prima che una cartella di lavoro venga salvata. Questo evento si verifica quando selezionate File ➪ Salva o File ➪ Salva con nome.

La seguente routine, che si trova nella finestra Codice per un oggetto Questa_cartella_di_lavoro, dimostra il comportamento dell'evento BeforeSave. La routine aggiorna il valore di una cella (la cella A1 in Foglio1) ogni volta che viene salvata la cartella di lavoro. In altre parole, la cella A1 funge da contatore per tenere traccia del numero di volte che il file è stato salvato.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _
     Cancel As Boolean)
     Dim Counter As Range
     Set Counter = Sheets("Foglio1").Range("A1")
     Counter.Value = Counter.Value + 1
End Sub
```

Notate che la routine Workbook_BeforeSave ha due argomenti: SaveAsUI e Cancel. Per dimostrare come funzionano questi argomenti, esaminate la seguente macro, che viene eseguita prima che la cartella di lavoro venga salvata. Questa routine tenta di impedire all'utente di salvare la cartella di lavoro con un nome diverso. Se l'utente sceglie File ➪ Salva con nome, l'argomento SaveAsUI sarà True.

Quando viene eseguito il codice, controlla il valore SaveAsUI. Se questa variabile è True, la routine visualizza un messaggio e imposta Cancel su True, cosa che annulla l'operazione di salvataggio.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI _
     As Boolean, Cancel As Boolean)
     If SaveAsUI Then
         MsgBox "Non puoi salvare una copia di questa cartella!"
     Cancel = True
     End If
End Sub
```

Notate che questa routine non impedirà davvero di salvare una copia con un altro nome. Per farlo, basta aprire la cartella di lavoro con le macro disattivate. Quando le macro sono disattivate, anche le routine di gestione degli eventi sono disattivate, il che ha perfettamente senso perché, dopo tutto, sono macro.

Esempi di eventi di attivazione

Un'altra categoria di eventi consiste nell'attivazione e disattivazione di oggetti, in particolare, fogli e cartelle di lavoro.

Attivare e disattivare gli eventi in un foglio

Excel può rilevare quando un determinato foglio viene attivato o disattivato ed eseguire una macro quando si verifica uno di questi eventi. Queste routine di gestione degli eventi si trovano nella finestra Codice per l'oggetto Foglio.

CONSIGLIO

Potete accedere rapidamente alla finestra Codice di un foglio facendo clic destro sulla scheda del foglio e selezionando Visualizza codice.

L'esempio seguente mostra una semplice routine che viene eseguita ogni volta che viene attivato un determinato foglio. Questo codice presenta semplicemente una fastidiosa finestra contenente un messaggio che mostra il nome del foglio attivo:

```
Private Sub Worksheet_Activate()
     MsgBox "Hai appena attivato " & ActiveSheet.Name
End Sub
```

Ecco un altro esempio che attiva la cella A1 ogni volta che viene attivato il foglio:

```
Private Sub Worksheet_Activate()
     Range("A1").Activate
End Sub
```

Il codice in queste due routine è banale, ma il gestore di eventi può essere complesso a piacere.

La routine seguente (memorizzata nella finestra Codice per l'oggetto Foglio1) utilizza l'evento Deactivate per impedire a un utente di attivare qualsiasi altro foglio nella cartella di lavoro. Se l'utente disattiva il Foglio1 (ovvero, se attiva un altro foglio), riceve un messaggio e il Foglio1 viene riattivato.

```
Private Sub Worksheet_Deactivate()
     MsgBox "Devi rimanere su Foglio1"
     Sheets("Foglio1").Activate
End Sub
```

A questo proposito, non è una buona idea usare routine VBA come questa per tentare di mantenere il controllo di Excel. Queste applicazioni così “imperative” possono essere molto frustranti e fastidiose. E, naturalmente, possono essere facilmente aggirate disattivando le macro.

Attivare e disattivare gli eventi in una cartella di lavoro

Gli esempi precedenti utilizzano eventi associati a uno specifico foglio di lavoro. Ma l’oggetto Questa_cartella_di_lavoro gestisce anche gli eventi che riguardano l’attivazione e disattivazione dei fogli. La seguente routine, memorizzata nella finestra Codice dell’oggetto Questa_cartella_di_lavoro, viene eseguita quando viene attivato *un qualsiasi* foglio nella cartella di lavoro. Il codice visualizza un messaggio con il nome del foglio attivato.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
     MsgBox Sh.Name
End Sub
```

La routine Workbook_SheetActivate utilizza l’argomento Sh. Sh è una variabile che rappresenta l’oggetto Foglio attualmente attivo. La finestra visualizza la proprietà Name dell’oggetto Foglio.

Il prossimo esempio è contenuto in una finestra Codice di Questa_cartella_di_lavoro. Consiste di due routine di gestione degli eventi.

- **Workbook_SheetDeactivate**: viene eseguita quando viene disattivato qualsiasi foglio nella cartella di lavoro. Memorizza in una variabile oggetto il foglio che è stato disattivato, ma solo se è un foglio di lavoro. La parola riservata Set crea una variabile oggetto, che è disponibile per tutte le routine nel modulo.
- **Workbook_SheetActivate**: viene eseguita quando viene attivato qualsiasi foglio nella cartella di lavoro. Controlla il tipo di foglio attivato (usando la funzione TypeName). Se è un foglio grafico, l’utente riceve un messaggio (vedi la Figura 11.5). Quando fate clic sul pulsante OK nella finestra del messaggio, viene riattivato il foglio di lavoro *precedente* (che è stato memorizzato nella variabile OldSheet).

CONSIGLIO

Sul sito web di questo libro è disponibile una cartella di lavoro contenente la versione originale inglese di questo codice.

```
Dim OldSheet As Object

Private Sub Workbook_SheetDeactivate(ByVal Sh As Object)
     If TypeName(Sh) = "Worksheet" Then Set OldSheet = Sh
End Sub

Private Sub Workbook_SheetActivate(ByVal Sh As Object)
     Dim Msg As String
     If TypeName(Sh) = "Chart" Then
         Msg = "Questo grafico considera "
         Msg = Msg & ActiveChart.SeriesCollection(1).Points.Count
         Msg = Msg & " punti di dati." & vbNewLine
         Msg = Msg & "Fate clic su OK per tornare a " & OldSheet.Name
         MsgBox Msg
         OldSheet.Activate
     End If
End Sub
```

FIGURA 11.5 Quando viene attivato un foglio grafico, l'utente riceve un messaggio come questo.

Eventi di attivazione della cartella di lavoro

Excel riconosce anche l'evento che si verifica quando si attiva o disattiva una determinata cartella di lavoro. Il seguente codice, contenuto nella finestra Codice per l'oggetto Questa_cartella_di_lavoro, viene eseguito ogni volta che la cartella di lavoro viene attivata. La routine non fa altro che ingrandire la finestra della cartella di lavoro.

```
Private Sub Workbook_Activate()
     ActiveWindow.WindowState = xlMaximized
End Sub
```

Viene visualizzato di seguito un esempio di codice Workbook_Deactivate. Questa routine viene eseguita quando una cartella di lavoro viene disattivata e copia l'intervallo selezionato ogni volta che la cartella di lavoro viene disattivata. Potrebbe essere utile se si copiano dati da più aree e si incollano in un'altra cartella di lavoro. Quando questa routine si attiva, potete selezionare l'intervallo da copiare, attivare l'altra cartella di lavoro, selezionare la destinazione e premere Ctrl + V (o Invio) per incollarvi i dati copiati.

```
Private Sub Workbook_Deactivate()
     ThisWorkbook.Windows(1).RangeSelection.Copy
End Sub
```

Altri eventi relativi ai fogli di lavoro

Il paragrafo precedente ha illustrato esempi per gli eventi di attivazione e disattivazione del foglio di lavoro. In questa parte del capitolo troverete ulteriori eventi che si verificano nei fogli di lavoro: doppio clic su una cella, clic destro su una cella e modifica di una cella.

L'evento BeforeDoubleClick

Potete impostare una routine VBA da eseguire quando l'utente fa doppio clic su una cella. Nell'esempio seguente (che è memorizzato nella finestra Codice per un oggetto Foglio), facendo doppio clic su una cella del foglio, tale cella viene messa in grassetto (se non lo è) oppure viene eliminato il grassetto (se lo è):

```
Private Sub Worksheet_BeforeDoubleClick _
       (ByVal Target As Excel.Range, Cancel As Boolean)
     Target.Font.Bold = Not Target.Font.Bold
     Cancel = True
End Sub
```

La routine Worksheet_BeforeDoubleClick ha due argomenti: Target e Cancel. Target rappresenta la cella (un oggetto Range) sulla quale è stato fatto doppio clic. Se Cancel è impostato su True, l'azione di doppio clic predefinita non si verifica.

L'azione predefinita per un doppio clic in una cella pone Excel in modalità di modifica cella. In questo caso, quando si fa doppio clic sulle celle, non è necessario modificare la cella, quindi impostate Cancel su True.

L'evento BeforeRightClick

L'evento BeforeRightClick è simile all'evento BeforeDoubleClick, tranne per il fatto che si occupa del clic destro su una cella. La seguente routine controlla se la cella su cui si è fatto clic destro contiene un valore numerico. In tal caso, il codice visualizza la finestra di dialogo Format celle, con la scheda Numero e imposta l'argomento Cancel su True (evitando così la normale visualizzazione del menu contestuale). Se la cella non contiene un valore numerico, non accade nulla di speciale; il menu contestuale viene visualizzato come al solito.

```
Private Sub Worksheet_BeforeRightClick _
       (ByVal Target As Excel.Range, Cancel As Boolean)
     If IsNumeric(Target) And Not IsEmpty(Target) Then
         Application.CommandBars.ExecuteMso ("NumberFormatsDialog")
         Cancel = True
     End If
End Sub
```

CONSIGLIO

Notate che il codice, disponibile sul sito web di questo libro, esegue un controllo aggiuntivo per verificare che la cella non sia vuota. Questo perché VBA considera le celle vuote come numeriche.

L'evento Change

L'evento Change si verifica ogni volta che una cella del foglio di lavoro viene modificata. Nell'esempio seguente, la routine Worksheet_Change impedisce a un utente di immettere nella cella A1 un valore non numerico. Questo codice è memorizzato nella finestra Codice per un oggetto Foglio.

```
Private Sub Worksheet_Change(ByVal Target As Range)
     If Target.Address = "$A$1" Then
         If Not IsNumeric(Target) Then
              MsgBox "Inserire un numero nella cella A1."
              Range("A1").ClearContents
              Range("A1").Activate
         End If
     End If
End Sub
```

Il singolo argomento per la routine Worksheet_Change, Target, rappresenta l'intervallo che è stato modificato. La prima istruzione controlla se l'indirizzo della cella è $A$1. In tal caso, il codice utilizza la funzione IsNumeric per determinare se la cella contenga un valore numerico. In caso contrario, visualizza un messaggio e il valore della cella viene cancellato. Viene poi attivata la cella A1, utile se, dopo l'immissione, il cursore della cella viene spostato in un'altra cella. Se l'intervento riguarda qualsiasi altra cella, non A1, non succede nulla.

Perché non utilizzare la convalida dei dati?

È opportuno conoscere il comando Dati ➪ Strumenti dati ➪ Convalida dati, il quale semplifica l'inserimento di dati del tipo corretto in una determinata cella o intervallo. Sebbene Dati ➪ Strumenti dati ➪ Convalida dati sia utile, non è però infallibile.

Provate ad aggiungere la convalida dei dati a una cella. Per esempio, potete impostarla in modo che la cella accetti solo un valore numerico. Funziona bene fino a quando non si copia un'altra cella e la si incolla nella cella convalidata. L'operazione di Incolla rimuove la convalida dei dati. È come se non ci fosse mai stata. La gravità di questo difetto dipende dalla vostra applicazione.

PER I PIÙ CURIOSI

Il comando Incolla elimina la convalida dei dati perché Excel considera la convalida un formato della cella. Pertanto, ha la stessa "dignità" della dimensione del carattere, del colore o di altri attributi simili. Quando si incolla qualcosa in una cella, si sostituiscono i formati nella cella di destinazione con quelli della cella di origine. Sfortunatamente, questi formati includono anche le regole di convalida dei dati.

Prevenire l'eliminazione della convalida dei dati

La routine di questo paragrafo mostra come sia possibile impedire agli utenti di copiare i dati e cancellare così le regole di convalida dei dati. In questo esempio si presuppone che il foglio di lavoro abbia un intervallo denominato InputArea e che quest'area di input contenga regole di convalida dei dati (impostate scegliendo Dati ➪ Strumenti dati ➪ Convalida dati). L'intervallo può avere qualsiasi regola di convalida necessaria.

CONSIGLIO

Sul sito web di questo libro è disponibile una cartella di lavoro contenente la versione originale inglese di questo codice.

```
Private Sub Worksheet_Change(ByVal Target As Range)
     Dim VT As Long
     'Le celle nell'intervallo di convalida
     'hanno ancora la convalida?
     On Error Resume Next
     VT = Range("InputRange").Validation.Type
     If Err.Number <> 0 Then
         Application.Undo
         MsgBox "L'ultima operazione è stata annullata, in quanto" & _
         " eliminerebbe le regole di convalida dei dati.", vbCritical
     End If
End Sub
```

La routine viene eseguita ogni volta che viene modificata una cella. Controlla il tipo di convalida dell'intervallo (denominato InputRange) che *dovrebbe* contenere le regole di convalida dei dati. Se la variabile VT contiene un errore, una o più celle di InputRange non hanno più la convalida dei dati (l'utente vi ha probabilmente copiato alcuni dati sopra). In questo caso, il codice esegue il metodo Undo dell'oggetto Application e inverte l'azione dell'utente. Quindi visualizza un messaggio, come illustrato nella Figura 11.6. Per ulteriori informazioni sull'utilizzo di On Error Resume Next consultate il Capitolo 12.

FIGURA 11.6 Esecuzione della convalida dei dati con una routine per eventi.

Il risultato? Diventa impossibile cancellare le regole di convalida copiando i dati. Quando qualcuno prova a danneggiare i dati, a volte potete utilizzare il linguaggio VBA per ripararli.

Eventi non associati agli oggetti

Gli eventi trattati in precedenza in questo capitolo sono associati a un oggetto cartella di lavoro o foglio di lavoro. Questo paragrafo si concentra su due tipi di eventi che non sono associati ad alcun oggetto: il tempo e la pressione dei tasti.

RICORDA!

Poiché il tempo e la pressione dei tasti non sono associati a un oggetto particolare, come una cartella di lavoro o un foglio di lavoro, questi gestori di eventi vengono programmati in un normale modulo di codice VBA (a differenza

degli altri gestori di eventi trattati in questo capitolo).

L'evento OnTime

L'evento OnTime si verifica a una determinata ora del giorno. Nell'esempio seguente viene illustrato come far eseguire a Excel una routine quando si verifica un evento alle 15:00. In questo caso, una voce robotica vi dà la sveglia, accompagnando la voce con una finestra contenente un messaggio:

```
Sub SetAlarm()
     Application.OnTime 0.625, "DisplayAlarm"
End Sub

Sub DisplayAlarm()
     Application.Speech.Speak ("Ehi, sveglia!")
     MsgBox " E' il momento del break pomeridiano!"
End Sub
```

In questo esempio, utilizzate il metodo OnTime dell'oggetto Application. Questo metodo accetta due argomenti: il tempo (0.625, ovvero le 15:00) e il nome della routine Sub da eseguire quando si verifica l'evento temporale (DisplayAlarm).

Questa routine è molto utile se tendete a essere troppo impegnati sul lavoro, tanto da dimenticare le riunioni e gli appuntamenti. Basta impostare un evento OnTime per ricordarvelo.

CONSIGLIO

La maggior parte delle persone ha difficoltà a pensare al tempo nei termini di Excel. Pertanto, potete utilizzare la funzione TimeValue per rappresentare l'ora. TimeValue converte una stringa che somiglia a un valore temporale in un valore che possa essere gestito da Excel. La seguente istruzione mostra un modo più semplice per programmare un evento per le 15:00 (le ore 3 p.m.):

```
Application.OnTime TimeValue("3:00:00 pm"), "DisplayAlarm"
```

Se desiderate pianificare un evento relativo all'ora corrente (per esempio, a 20 minuti da ora), potete utilizzare un'istruzione come questa:

```
Application.OnTime Now + TimeValue("00:20:00"), "DisplayAlarm"
```

Inoltre potreste utilizzare il metodo OnTime per eseguire una routine VBA in un particolare giorno. Dovete però assicurarvi che il computer sia in funzione e che la cartella di lavoro con la routine sia mantenuta aperta. La seguente istruzione esegue la routine DisplayAlarm alle 17:00 del 31 dicembre 2019:

```
Application.OnTime DateValue("12/31/2019 5:00 pm"), "DisplayAlarm"
```

Questa particolare riga di codice potrebbe tornare utile per avvertirvi che dovete andare a casa e prepararvi per il cenone di Capodanno.

Ecco un altro esempio che utilizza l'evento OnTime. L'esecuzione delle routine UpdateClock scrive l'ora sulla cella A1 e programma anche un altro evento cinque secondi dopo. Questo evento riesegue la routine UpdateClock. L'effetto netto è che la cella A1 viene aggiornata con l'ora corrente ogni cinque secondi. Per interrompere gli eventi, eseguite la routine StopClock (che annulla l'evento). Notate che NextTick è una variabile a livello di modulo che memorizza l'ora per l'evento successivo.

```
Dim NextTick As Date

Sub UpdateClock()
'   Aggiorna la cella A1 con l'ora corrente
     ThisWorkbook.Sheets(1).Range("A1") = Time
'   Imposta il prossimo evento a cinque secondi da ora
     NextTick = Now + TimeValue("00:00:05")
     Application.OnTime NextTick, "UpdateClock"
End Sub

Sub StopClock()
'   Annulla l'evento OnTime (ferma l'orologio)
     On Error Resume Next
     Application.OnTime NextTick, "UpdateClock", , False
End Sub
```

WARNING

L'evento OnTime persiste anche dopo la chiusura della cartella di lavoro. In altre parole, se chiudete la cartella di lavoro senza eseguire la routine StopClock, la cartella di lavoro si riapre cinque secondi dopo (presupponendo che Excel sia ancora in esecuzione). Per evitare ciò, utilizzate una routine di evento Workbook_BeforeClose che contenga la seguente istruzione:

```
Call StopClock
```

Il metodo OnTime ha due argomenti aggiuntivi. Se prevedete di utilizzare questo metodo, fate riferimento alla Guida per tutti i dettagli.

Il metodo OnTime ha un moltissimi utilizzi per tutti i tipi di applicazione. La Figura 11.7 illustra un orologio analogico che utilizza il metodo OnTime per muovere la lancetta dei secondi. Il quadrante dell'orologio è in realtà un grafico, che viene aggiornato ogni secondo per visualizzare l'ora.

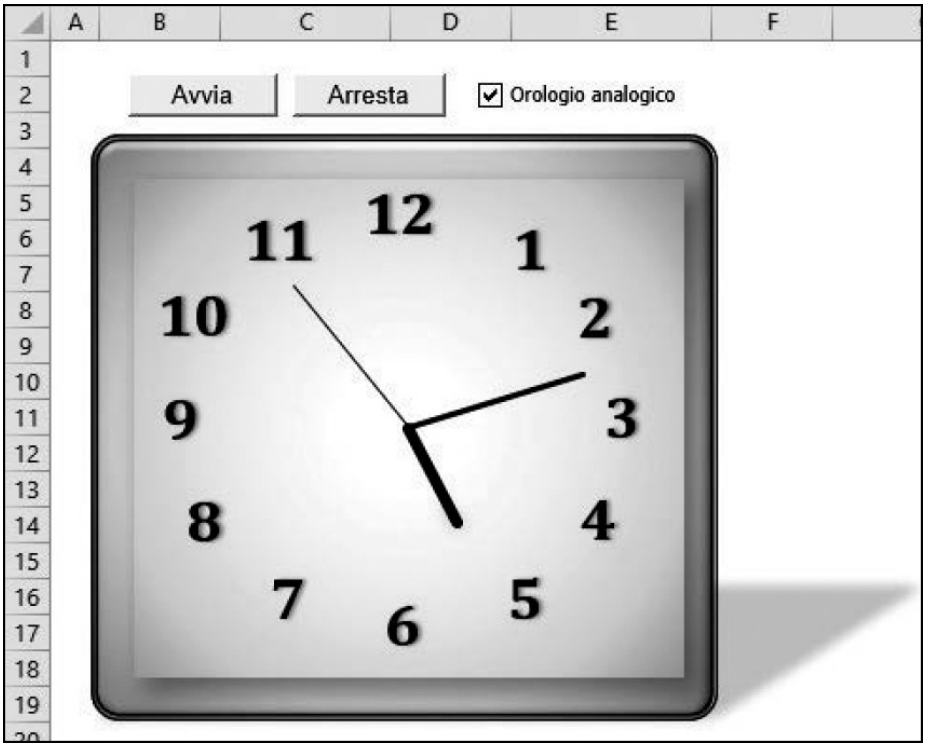

FIGURA 11.7 Un esempio di orologio analogico.

Eventi per pressioni di tasti

Excel controlla costantemente quello che scrivete. Per questo motivo, potete impostare le cose in modo che una certa combinazione di tasti lanci una routine.

Ecco un esempio che riassegna i tasti PgDn e PgUp:

```
Sub Setup_OnKey()
     Application.OnKey "{PgDn}", "PgDn_Sub"
     Application.OnKey "{PgUp}", "PgUp_Sub"
End Sub

Sub PgDn_Sub()
     On Error Resume Next
         ActiveCell.Offset(1, 0).Activate
End Sub

Sub PgUp_Sub()
     On Error Resume Next
         ActiveCell.Offset(-1, 0).Activate
End Sub
```

Dopo aver impostato gli eventi OnKey eseguendo la routine Setup_OnKey, premendo PgDn vi sposterete di una riga verso il basso. Premendo PgUp vi sposterete di una riga verso l'alto.

Notate che i codici dei tasti sono racchiusi tra parentesi graffe, non tra normali parentesi. Per un elenco completo dei codici dei tasti, consultate la Guida. Cercate *OnKey*.

In questo esempio, On Error Resume Next ignora eventuali errori generati. Per esempio, se la cella attiva si trova nella prima riga, provare a spostarsi verso l'alto di una riga provoca un errore, che può essere tranquillamente ignorato. E se vi trovate in un foglio grafico, non esiste il concetto di cella attiva.

Eseguendo la seguente routine, annullate gli eventi OnKey:

```
Sub Cancel_OnKey()
     Application.OnKey "{PgDn}"
     Application.OnKey "{PgUp}"
End Sub
```

WARNING

L'utilizzo di una stringa vuota come secondo argomento per il metodo OnKey non annulla l'evento OnKey. Piuttosto, fa sì che Excel ignori il tasto. Per esempio, la seguente istruzione chiede a Excel di ignorare la combinazione di tasti Alt + F4. Il segno di percentuale rappresenta il tasto Alt:

```
Application.OnKey "%{F4}", ""
```

Sebbene sia possibile utilizzare il metodo OnKey per assegnare un tasto di scelta rapida per l'esecuzione di una macro, per questa attività è necessario utilizzare la finestra di dialogo Opzioni macro. Per maggiori dettagli, consultate il Capitolo 5.

RICORDA!

Se chiudete la cartella di lavoro che contiene il codice e lasciate aperto Excel, il metodo OnKey non viene reimpostato. Di conseguenza, premendo il tasto di scelta rapida Excel apre automaticamente il file con la macro. Per evitare che ciò accada, includete del codice nell'evento Workbook_BeforeClose (mostrato in precedenza in questo capitolo) per reimpostare l'evento OnKey.

Capitolo 12

Tecniche di gestione degli errori

IN QUESTO CAPITOLO

- **Capire la differenza tra errori di programmazione ed errori runtime**
- **Ricercare e gestire gli errori runtime**
- **Usare le istruzioni On Error e Resume**
- **Scoprire come si può sfruttare un errore a proprio vantaggio**

Errare è umano, dicono. Anticipare gli errori è divino. Quando si lavora con il linguaggio VBA, è necessario essere a conoscenza dell'esistenza di due grandi classi di errori: gli errori di programmazione e gli errori in esecuzione, in gergo "errori *runtime*". Questo capitolo è interamente dedicato agli errori *runtime*.

Un programma ben scritto gestisce gli errori come Fred Astaire: con eleganza. Fortunatamente, VBA include diversi strumenti per aiutarvi a identificare gli errori e gestirli con garbo.

Tipi di errori

Se avete provato uno degli esempi di questo libro, probabilmente avrete ricevuto uno o più messaggi di errore. Alcuni di questi errori derivano dalla scrittura di codice VBA errato. Per esempio, potete aver digitato una parola riservata in maniera errata o un'istruzione con la sintassi errata. Se commettete un errore di questo tipo, non sarete nemmeno in grado di eseguire la routine, finché non lo correggerete.

Questo capitolo non tratta questi tipi di errori. Si concentra, invece, sugli errori runtime: gli errori che si verificano mentre Excel esegue il codice VBA. Per la precisione, questo capitolo tratta i seguenti affascinanti argomenti:

- come identificare gli errori;
- che cosa fare quando si verifica un errore;
- come ripristinare le cose dopo un errore;
- come creare errori intenzionali (sì, a volte un errore può anche essere una buona cosa).

L'obiettivo finale della gestione degli errori consiste nel predisporre del codice che eviti il più possibile di visualizzare all'utente un messaggio d'errore di Excel. In altre parole, desiderate anticipare i potenziali errori e gestirli prima che Excel abbia la possibilità di affliggervi con un messaggio d'errore, solitamente tutto tranne che informativo.

Un esempio errato

Per iniziare, date un'occhiata alla seguente macro. Una macro così semplice non potrebbe mai produrre errori, giusto?

Attivate il VBE, create un nuovo modulo e inseritevi il seguente codice:

```
Sub EnterSquareRoot()
     Dim Num As Double
'   Chiede un valore
     Num = InputBox("Inserite un valore")
'   Inserisce la radice quadrata
     ActiveCell.Value = Sqr(Num)
End Sub
```

Come potete vedere nella Figura 12.1, questa routine richiede all'utente un valore. Poi esegue un calcolo "magico" e inserisce nella cella attiva la radice quadrata di quel valore.

CONSIGLIO

Potete eseguire questa routine direttamente dal VBE, premendo F5. In alternativa, potreste voler aggiungere a un foglio di lavoro un pulsante (per eseguire questa operazione, selezionate il comando Sviluppo ➪ Controlli ➪ Inserisci e selezionate Pulsante nel gruppo Controlli modulo; poi assegnate la macro al pulsante (Excel richiede la macro da assegnare). Quindi potrete lanciare la routine semplicemente facendo clic sul pulsante.

FIGURA 12.1 La funzione InputBox apre una finestra di dialogo che richiede all'utente un valore.

La macro non è proprio perfetta

Eseguite il codice un paio di volte per provarlo. Funziona piuttosto bene, vero? Ora, quando vi viene richiesto un valore, provate a inserire un numero negativo. Ops! Cercare di calcolare la radice quadrata di un numero negativo è illegale su questo pianeta.

Excel risponde alla richiesta di calcolo della radice quadrata di un numero negativo visualizzando il messaggio d'errore runtime rappresentato nella Figura 12.2. Per ora, basta fare clic sul pulsante Fine. Se fate clic sul pulsante Debug, Excel sospende la macro in modo da poter utilizzare gli strumenti di debug per aiutare a rintracciare l'errore (gli strumenti di debug sono trattati nel Capitolo 13).

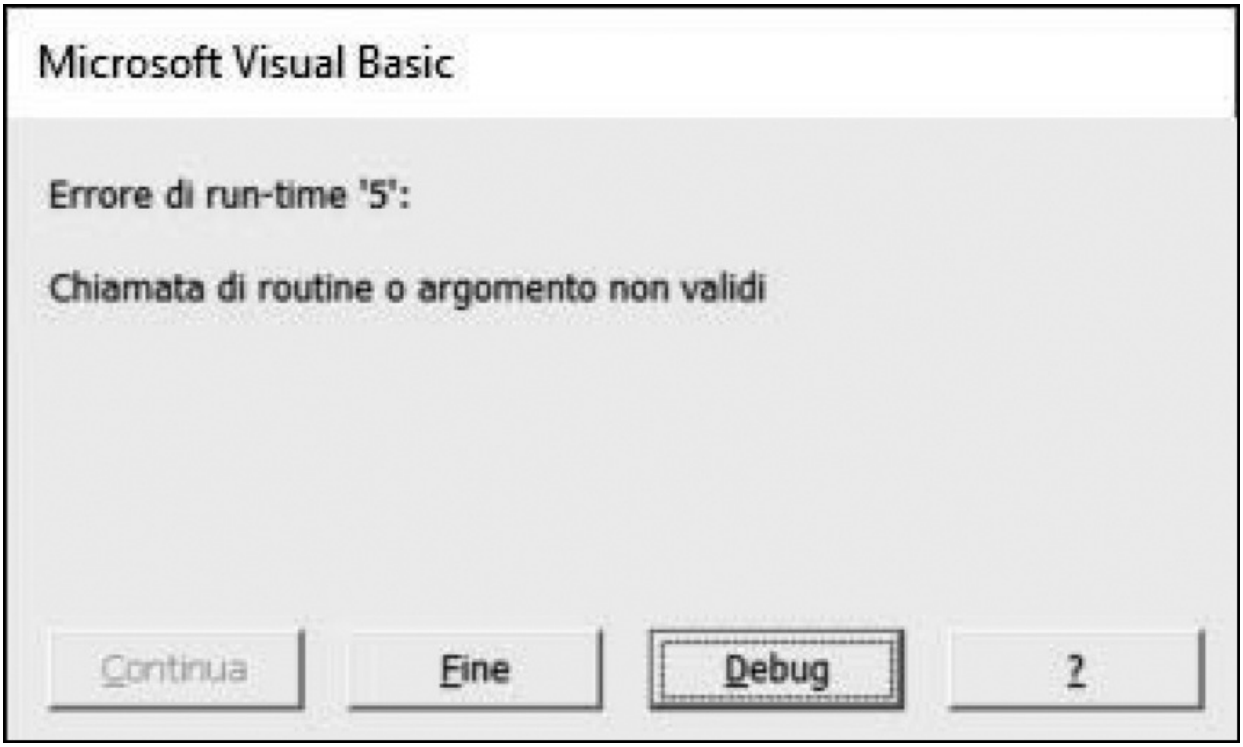

FIGURA 12.2 Excel visualizza questo messaggio d'errore quando la routine tenta di calcolare la radice quadrata di un numero negativo.

La maggior parte delle persone non trova particolarmente utili i messaggi di errore di Excel (per esempio, che cosa significa “Chiamata di routine o argomento non validi”?). Per migliorare la routine, è necessario anticipare questo errore e gestirlo in modo più elegante. In altre parole, è necessario aggiungere alla routine del codice di gestione degli errori.

Ecco una versione modificata di EnterSquareRoot:

```
Sub EnterSquareRoot2()
     Dim Num As Double
'   Chiede un valore
     Num = InputBox("Inserite un valore")

'   Si assicura che il numero non sia negativo
     If Num < 0 Then
         MsgBox "Dovete inserire un numero positivo."
         Exit Sub
     End If

'   Inserisce la radice quadrata
     ActiveCell.Value = Sqr(Num)
End Sub
```

Una struttura If-Then controlla il valore contenuto nella variabile Num. Se Num è minore di 0, la routine apre una finestra contenente informazioni effettivamente comprensibili ai comuni mortali. La routine termina poi con l’istruzione Exit Sub, quindi l’errore runtime non ha mai la possibilità di verificarsi.

La macro non è ancora perfetta

Quindi ora la routine EnterSquareRoot modificata è perfetta, giusto? Non proprio. Provate a inserire del testo anziché un valore. Oppure fate clic sul pulsante Annulla nella casella di inserimento. Entrambe queste azioni generano un errore (*Tipo non corrispondente*). Questa semplice piccola routine richiede pertanto ulteriore codice di gestione degli errori.

Il seguente codice modificato utilizza la funzione IsNumeric per assicurarsi che Num contenga un valore numerico. Se l’utente immette qualcosa di diverso da un numero, la routine visualizza un messaggio e poi termina. Inoltre, notate che la variabile Num è ora dichiarata come Variant. Se fosse stata dichiarata come Double, e l’utente avesse inserito nella casella di input un valore non numerico, il codice avrebbe generato un errore non gestito.

```
Sub EnterSquareRoot3()
     Dim Num As Variant
'   Chiede un valore
     Num = InputBox("Inserite un valore")

'   Si assicura che Num sia un numero
     If Not IsNumeric(Num) Then
         MsgBox "Dovete inserire un valore numerico."
         Exit Sub
     End If

'   Si assicura che il numero non sia negativo
     If Num < 0 Then
         MsgBox "Dovete inserire un numero positivo."
         Exit Sub
     End If

'   Inserisce la radice quadrata
     ActiveCell.Value = Sqr(Num)
End Sub
```

La macro ora è perfetta, vero?

Ora questo codice è assolutamente perfetto, giusto? Non proprio. Provate a eseguire la routine mentre è attivo un foglio grafico. Eccolo là: un altro errore runtime, questa volta il temuto numero 91 (vedi la Figura 12.3). Questo errore si verifica quando è attivo un foglio grafico, e non c’è nessuna cella da attivare o comunque quando è selezionato qualcosa di diverso da un intervallo.

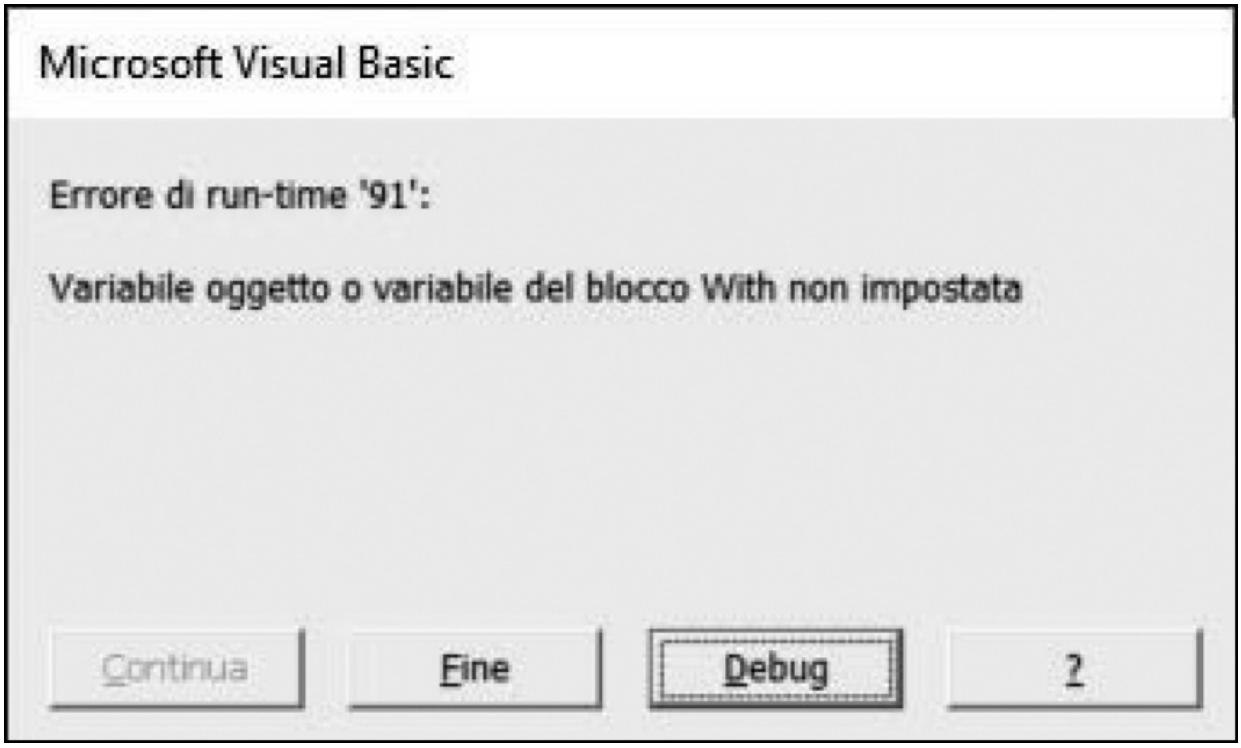

FIGURA 12.3 L'esecuzione della routine quando è selezionato un foglio grafico genera questo errore.

La seguente routine usa la funzione TypeName per assicurarsi che la selezione sia un intervallo. Se viene selezionato qualcosa di diverso da un intervallo, questa routine visualizza un messaggio e poi esce:

```
Sub EnterSquareRoot4()
     Dim Num As Variant
'   Si assicura che sia attivo un foglio di lavoro
     If TypeName(Selection) <> "Range" Then
         MsgBox "Selezionate la cella per il risultato."
         Exit Sub
     End If

'   Chiede un valore
     Num = InputBox("Inserite un valore")
'   Si assicura che Num sia un numero
     If Not IsNumeric(Num) Then
         MsgBox "Dovete inserire un valore numerico."
         Exit Sub
     End If

'   Si assicura che il numero non sia negativo
     If Num < 0 Then
         MsgBox "Dovete inserire un numero positivo."
         Exit Sub
     End If

'   Inserisce la radice quadrata
     ActiveCell.Value = Sqr(Num)
End Sub
```

Rinunciare alla perfezione

Ormai, questa routine *deve sicuramente* essere perfetta. Ehm… non ancora.

Proteggete il foglio di lavoro (Revisione ➪ Proteggi ➪ Proteggi foglio) e poi eseguite il codice. Sì, il tentativo di scrivere su un foglio di lavoro protetto genera ancora un errore. E ci sono probabilmente decine di altri errori, impossibili da prevedere. Continuate a leggere e scoprirete che esiste un altro modo per affrontare gli errori, anche quelli imprevedibili.

Gestire gli errori in un altro modo

Come è possibile identificare e gestire ogni possibile errore? Spesso, la cosa, semplicemente, non è possibile. Fortunatamente, VBA offre un altro modo per gestire gli errori.

Revisione della routine EnterSquareRoot

Il seguente codice è una versione modificata della routine del paragrafo precedente. Qui, viene usata un'istruzione On Error che ricerca tutti gli errori e poi verifica se l'InputBox è stata cancellata.

```
Sub EnterSquareRoot5()
     Dim Num As Variant
     Dim Msg As String

'   Imposta la gestione degli errori
     On Error GoTo BadEntry
'   Chiede un valore
     Num = InputBox("Inserite un valore")

'   Esce in caso di Annulla
     If Num = "" Then Exit Sub

'   Inserisce la radice quadrata
     ActiveCell.Value = Sqr(Num)
     Exit Sub

BadEntry:
     Msg = "Si è verificato un errore." & vbNewLine & vbNewLine
     Msg = Msg & "Assicuratevi che sia selezionato un intervallo, "
     Msg = Msg & "che il foglio di lavoro non sia protetto "
     Msg = Msg & "e che il valore non sia negativo."
     MsgBox Msg, vbCritical
End Sub
```

Questa routine intercetta *qualsiasi* tipo di errore runtime. Dopo aver intercettato un errore runtime, la routine EnterSquareRoot modificata visualizza la finestra MsgBox rappresentata nella Figura 12.4. Questa finestra descrive le cause più probabili dell'errore.

FIGURA 12.4 Un errore runtime nella routine genera questo messaggio d'errore, non del tutto inutile.

CONSIGLIO

ON ERROR NON FUNZIONA?

Se un'istruzione On Error non funziona come dovrebbe, dovete modificare una delle impostazioni.

1. **Attivate il VBE.**
2. **Scegliete Strumenti ➪ Opzioni.**
3. **Fate clic sulla scheda Generale della finestra di dialogo Opzioni.**
4. **Assicuratevi che l'impostazione Interrompi ad ogni errore sia deselezionata.**

Se questa impostazione è selezionata, quello che accade è che Excel ignora essenzialmente le istruzioni On Error. Normalmente si desidera mantenere l'opzione Intercettazione degli errori impostata su Interrompi ad ogni errore non gestito.

Informazioni sull'istruzione On Error

L'uso di un'istruzione On Error nel codice VBA consente di aggirare la gestione degli errori incorporata di Excel e di utilizzare il codice di gestione degli errori che voi avete predisposto. Nell'esempio precedente, un errore runtime fa sì che l'esecuzione della macro passi all'istruzione con l'etichetta BadEntry. Di conseguenza, aggirate i messaggi di errore ostili di Excel e potrete visualizzare all'utente il messaggio che desiderate.

CONSIGLIO

Notate che nell'esempio viene utilizzata un'istruzione Exit Sub subito prima dell'etichetta BadEntry. Questa istruzione è necessaria perché non desiderate eseguire il codice di gestione degli errori anche quando *non* si verifica un errore.

Gestione degli errori: i dettagli

Potete utilizzare l'istruzione On Error in tre modi, come mostrato nella Tabella 12.1.

TABELLA 12.1 Utilizzo dell'istruzione On Error

Sintassi	Che cosa fa
On Error GoTo *etichetta*	Dopo aver eseguito questa istruzione, VBA riprende l'esecuzione dall'istruzione successiva all'*etichetta* specificata. Dopo l'etichetta è necessario includere i due punti, in modo che VBA la riconosca come un'etichetta.
On Error GoTo 0	Dopo aver eseguito questa istruzione, VBA riprende il normale comportamento di controllo degli errori. Utilizzate questa istruzione dopo aver utilizzato una delle altre istruzioni On Error o quando desiderate rimuovere la gestione degli errori dalla routine.
On Error Resume Next	Dopo aver eseguito questa istruzione, VBA ignora semplicemente tutti gli errori e riprende l'esecuzione dall'istruzione successiva.

Recupero dopo un errore

In alcuni casi, quando si verifica un errore desiderate semplicemente che la routine termini con garbo. Per esempio, potete visualizzare un messaggio che descrive l'errore e poi uscire dalla routine (l'esempio EnterSquareRoot5 mostrato in precedenza utilizza questa tecnica). In altri casi, se possibile, desiderate recuperare l'esecuzione dopo l'errore.

Per eseguire un recupero dopo un errore, è necessario utilizzare un'istruzione Resume. Questa cancella la condizione di errore e consente di continuare l'esecuzione, in qualche luogo. Potete utilizzare l'istruzione Resume in tre modi, come mostrato nella Tabella 12.2.

TABELLA 12.2 Utilizzo dell'istruzione Resume

Sintassi	Che cosa fa
Resume	L'esecuzione riprende con l'istruzione che ha causato l'errore. Utilizzatela solo se il codice di gestione degli errori ha corretto il problema ed è corretto continuare.
Resume Next	L'esecuzione riprende con l'istruzione immediatamente successiva all'istruzione che ha causato l'errore. Questo essenzialmente ignora l'errore.
Resume *etichetta*	L'esecuzione riprende dall'etichetta specificata.

L'esempio seguente utilizza un'istruzione Resume dopo che si è verificato un errore:

```
Sub EnterSquareRoot6()
     Dim Num As Variant
     Dim Msg As String
     Dim Ans As Integer


TryAgain:
'   Imposta la gestione degli errori
```

```
      On Error GoTo BadEntry

'   Chiede un valore
      Num = InputBox("Inserite un valore")
      If Num = "" Then Exit Sub

'   Inserisce la radice quadrata
      ActiveCell.Value = Sqr(Num)

      Exit Sub

BadEntry:
      Msg = Err.Number & ": " & Error(Err.Number)
      Msg = Msg & vbNewLine & vbNewLine
      Msg = Msg & "Assicuratevi che sia selezionato un intervallo, "
      Msg = Msg & "che il foglio di lavoro non sia protetto "
      Msg = Msg & "e che il valore non sia negativo."
      Msg = Msg & vbNewLine & vbNewLine & "Volete riprovare"
      Ans = MsgBox(Msg, vbYesNo + vbCritical)
      If Ans = vbYes Then Resume TryAgain
End Sub
```

Questa routine ha un'altra etichetta: TryAgain. Se si verifica un errore, l'esecuzione continua con l'etichetta BadEntry e il codice visualizza il messaggio rappresentato nella Figura 12.5. Se l'utente risponde facendo clic su Sì, viene attivata l'istruzione Resume e l'esecuzione torna all'etichetta TryAgain. Se l'utente fa clic su No, la routine termina.

FIGURA 12.5 Se si verifica un errore, l'utente può decidere se riprovare.

Notate che il messaggio d'errore include anche il codice d'errore, oltre alla descrizione.

L'istruzione Resume cancella, prima di procedere, la condizione di errore. Per capire che cosa significa, provate a sostituire, nell'esempio precedente, la seguente istruzione per la penultima istruzione:

```
If Ans = vbYes Then GoTo TryAgain
```

Il codice non funziona correttamente se utilizzate GoTo anziché Resume. Per dimostrarlo, inserite un numero negativo. Otterrete la finestra d'errore che avete predisposto. Ora fate clic su Sì per riprovare, e inserite *un altro* numero negativo. Questo secondo errore non viene bloccato, perché la condizione di errore originale non è stata annullata.

CONSIGLIO

Questo esempio è disponibile, in originale inglese, sul sito web del libro.

La gestione degli errori in breve

Per aiutarvi a venire a capo di tutta questa attività di gestione degli errori, ecco un rapido riassunto. Un blocco di

codice di gestione degli errori ha le seguenti caratteristiche.

- Inizia immediatamente dopo l'etichetta specificata nell'istruzione On Error.
- Dovrebbe essere raggiunto dalla macro solo se si verifica un errore. Ciò significa che è necessario utilizzare un'istruzione come Exit Sub o Exit Function immediatamente prima dell'etichetta.
- Potrebbe richiedere un'istruzione Resume. Se, quando si verifica un errore, scegliete di non interrompere la routine, dovrete eseguire un'istruzione Resume prima di tornare al codice principale.

Sapere quando ignorare gli errori

In alcuni casi, è perfettamente corretto ignorare gli errori. Ecco quando entra in gioco l'istruzione On Error Resume Next.

L'esempio seguente esamina ogni cella dell'intervallo selezionato e ne converte il valore nella sua radice quadrata. Questa routine genera un messaggio d'errore se una cella nella selezione contiene un numero negativo o un testo:

```
Sub SelectionSqrt()
     Dim cell As Range
     If TypeName(Selection) <> "Range" Then Exit Sub
     For Each cell In Selection
         cell.Value = Sqr(cell.Value)
     Next cell
End Sub
```

In questo caso, potete tranquillamente saltare qualsiasi cella contenente un valore che non è possibile convertire in una radice quadrata. Potete creare ogni sorta di controllo degli errori utilizzando strutture If-Then, ma cercate di escogitare la soluzione migliore (e più semplice) che consiste nell'ignorare, semplicemente, determinati errori.

La seguente routine lo fa sfruttando l'istruzione On Error Resume Next:

```
Sub SelectionSqrt()
     Dim cell As Range
     If TypeName(Selection) <> "Range" Then Exit Sub
     On Error Resume Next
     For Each cell In Selection
         cell.Value = Sqr(cell.Value)
     Next cell
End Sub
```

In generale, potete utilizzare un'istruzione On Error Resume Next quando avete a che fare con errori innocui o insignificanti.

Identificazione di specifici errori

Non tutti gli errori sono uguali. Alcuni sono gravi, altri meno. Se da un lato potete ignorare quelli che considerate irrilevanti, dovete assolutamente affrontare quelli più gravi. In alcuni casi, è necessario identificare l'errore specifico che si è verificato.

Ogni tipo di errore ha un suo codice. Quando si verifica un errore, Excel memorizza il codice dell'errore in un oggetto Error, di nome Err. La proprietà Number di questo oggetto contiene il codice dell'errore; la sua proprietà Description contiene una descrizione dell'errore. La seguente istruzione, per esempio, visualizza il codice dell'errore, i due punti e la descrizione dell'errore:

```
MsgBox Err.Number & ": " & Err.Description
```

La Figura 12.5 precedente mostra un esempio. Tenete presente, tuttavia, che i messaggi d'errore di Excel non sono sempre utilissimi… ma questo già lo sapete.

La seguente routine mostra come determinare quale errore si è verificato. In questo caso, potete tranquillamente ignorare gli errori causati dal tentativo di ottenere la radice quadrata di un numero non positivo (ovvero l'errore 5) o gli errori causati dal tentativo di ottenere la radice quadrata di un valore non numerico (errore 13). D'altra parte, dovete informare l'utente se il foglio di lavoro è protetto o la selezione contiene una o più celle bloccate (altrimenti, l'utente potrebbe pensare che la macro abbia funzionato quando in realtà non è vero). Tentare di scrivere su una cella bloccata in un foglio di lavoro protetto provoca l'errore con codice 1004.

```
Sub SelectionSqrt()
     Dim cell As Range
     Dim ErrMsg As String
     If TypeName(Selection) <> "Range" Then Exit Sub
     On Error GoTo ErrorHandler
     For Each cell In Selection
         cell.Value = Sqr(cell.Value)
     Next cell
     Exit Sub

ErrorHandler:
```

```
     Select Case Err.Number
         Case 5 'Numero negativo
              Resume Next
         Case 13 'Errore di tipo
              Resume Next
         Case 1004 'Cella bloccata, foglio protetto
              MsgBox "Cella bloccata. Riprovate.", vbCritical, cell.Address
              Exit Sub
         Case Else
              ErrMsg = Error(Err.Number)
              MsgBox "ERRORE: " & ErrMsg, vbCritical, cell.Address
              Exit Sub
     End Select
End Sub
```

Quando si verifica un errore runtime, l'esecuzione salta al codice che inizia con l'etichetta ErrorHandler. La struttura Select Case (discussa nel Capitolo 10) verifica tre codici d'errore. Se il codice d'errore è 5 o 13, l'esecuzione riprende dall'istruzione successiva (in altre parole, l'errore viene ignorato). Ma se il codice d'errore è 1004, la routine avvisa l'utente e poi si chiude. L'ultimo Case, un *passepartout* per gli errori imprevisti, intercetta tutti gli altri errori e visualizza il messaggio d'errore.

Un errore intenzionale

A volte, potete usare un errore a vostro vantaggio. Supponiamo di avere una macro che funziona solo se è aperta una determinata cartella di lavoro. Come potete determinare se tale cartella di lavoro è aperta? Un modo consiste nello scrivere del codice che esamina la collezione Workbooks, per determinare se contiene la cartella di lavoro che vi interessa.

Ecco un modo più semplice: utilizzare una funzione generica che accetta un argomento (un nome di cartella di lavoro) e restituisce True se la cartella di lavoro è aperta o False se non lo è.

Ecco la funzione:

```
Function WorkbookIsOpen(book As String) As Boolean
     Dim WBName As String
     On Error GoTo NotOpen
     WBName = Workbooks(book).Name
     WorkbookIsOpen = True
     Exit Function
NotOpen:
     WorkbookIsOpen = False
End Function
```

Questa funzione sfrutta il fatto che Excel genera un errore se fate riferimento a una cartella di lavoro che non è aperta. La seguente istruzione, per esempio, genera un errore se la cartella di lavoro NomeCartella.xlsx non è aperta:

```
WBName = Workbooks("NomeCartella.xlsx").Name
```

Nella funzione WorkbookIsOpen, l'istruzione On Error indica a VBA di continuare la macro con l'istruzione NotOpen se si verifica un errore. Pertanto, un errore indica che la cartella di lavoro non è aperta e la funzione restituisce False. Se la cartella di lavoro è *aperta*, non si verifica alcun errore e la funzione restituisce True.

Ecco un'altra versione della funzione WorkbookIsOpen. Questa versione utilizza l'istruzione On Error Resume Next per ignorare l'errore. Ma il codice controlla la proprietà Number di Err. Se Err.Number è 0, non si è verificato alcun errore e la cartella di lavoro è aperta. Se Err.Number è qualsiasi altra cosa, significa che si è verificato un errore (e quindi la cartella di lavoro non è aperta).

```
Function WorkbookIsOpen(book) As Boolean
     Dim WBName As String
     On Error Resume Next
     WBName = Workbooks(book).Name
     If Err.Number = 0 Then WorkbookIsOpen = True _
       Else WorkbookIsOpen = False
End Function
```

L'esempio seguente mostra come utilizzare questa funzione in una routine Sub:

```
Sub UpdatePrices()
     If Not WorkbookIsOpen("Prezzi.xlsx") Then
         MsgBox "Aprite la cartella di lavoro Prezzi, prima!"
         Exit Sub
     End If
'   [Altro codice]
End Sub
```

La routine UpdatePrices (che deve trovarsi nella stessa cartella di lavoro di WorkbookIsOpen) richiama la funzione WorkbookIsOpen e le passa come argomento il nome della cartella di lavoro (Prezzi.xlsx). La funzione WorkbookIsOpen restituisce True o False. Pertanto, se la cartella di lavoro non è aperta, la routine informa l'utente di tale circostanza.

Se invece la cartella di lavoro è aperta, la macro continua l’esecuzione.

La gestione degli errori può essere un’idea complicata. Dopo tutto, si possono verificare molti errori e non potete prevederli tutti. In generale, dovreste intercettare gli errori e correggere, se possibile, la situazione prima che intervenga Excel. Scrivere un codice di ricerca degli errori efficace richiede una conoscenza approfondita di Excel e una chiara comprensione di come funziona la gestione degli errori VBA. I prossimi capitoli contengono altri esempi di gestione degli errori.

Capitolo 13

Tecniche di eliminazione dei bug

IN QUESTO CAPITOLO

- **Che cosa sono i bug e perché sterminarli è cosa buona e giusta**
- **I tipi di bug che affliggono i programmi**
- **Tecniche per il debug del codice**
- **Usare gli strumenti di debug integrati nel VBE**
- **Un elenco di suggerimenti pratici per il contenimento dei bug**

Se il termina *bug* evoca in voi solo l'immagine di un coniglio dei cartoni animati, questo capitolo vi chiarirà qualche concetto. In poche parole, un *bug* è un errore di programmazione. Qui vi parlerò di bug di programmazione: come identificarli e come eliminarli dal vostro modulo.

Vari tipi di bug

Come probabilmente sapete, un *bug* è un problema con il software. In altre parole, se il software non funziona come previsto, ha un *bug*. Il fatto è che tutti i principali programmi, anche quelli più blasonati, contengono *bug*, e anche molti. Excel stesso soffre di centinaia (se non migliaia) di *bug*. Fortunatamente, la stragrande maggioranza di questi è relativamente oscura e si manifesta solo in circostanze molto specifiche.

Nella scrittura di programmi VBA non banali, il vostro codice con ogni probabilità conterrà dei bug. Bisogna semplicemente accettarlo e non è necessariamente una conseguenza della vostra abilità di programmazione. I bug possono rientrare in una delle seguenti categorie.

- **Difetti logici nel codice:** spesso questi bug si possono evitare pensando con maggiore attenzione al problema affrontato dal vostro programma.
- **Bug per contesto errato:** questo tipo di bug si verifica quando tentate di fare qualcosa nel posto o nel momento sbagliato. Per esempio, il codice potrebbe provare a scrivere dei dati nelle celle nel foglio attivo, quando il foglio attivo è in realtà un foglio grafico (che non ha celle).
- **Bug per casi estremi:** questi bug si presentano quando si incontrano dati imprevisti, come numeri molto grandi o molto piccoli.
- **Bug per tipo di dati errato:** questo tipo di errore si verifica quando tentate di elaborare dati di tipo errato, come quando tentate di calcolare la radice quadrata di una stringa di testo.
- **Bug per versione errata:** questo tipo di bug fa riferimento alle incompatibilità tra diverse versioni di Excel. Per esempio, potete sviluppare una cartella di lavoro con Excel 2019, per poi scoprire che non funziona con Excel 2003. In genere, potete evitare tali problemi non usando funzionalità specifiche della versione. Spesso, l'approccio più semplice consiste nello sviluppare l'applicazione utilizzando il numero di versione di Excel più basso che gli utenti potrebbero avere. In ogni caso, tuttavia, dovreste provare il vostro lavoro su tutte le versioni che prevedete verranno utilizzate.
- **Bug indipendenti dal vostro controllo:** questi sono i più frustranti. Un esempio si ha quando Microsoft aggiorna Excel e apporta una piccola modifica non documentata che improvvisamente rende inutilizzabile la macro. È noto che anche gli aggiornamenti di sicurezza possono causare problemi.

Il *debug* è il processo di identificazione e correzione dei bug nel programma. Sviluppare capacità di debug richiede tempo, quindi non scoraggiatevi se all'inizio questo processo vi risulterà difficile.

RICORDA!

È importante capire la distinzione esistente tra *bug* ed *errori di sintassi*. Un errore di sintassi è un errore legato al

linguaggio VBA. Potreste sbagliare, per esempio, l'ortografia di una parola riservata, omettere l'istruzione Next in un ciclo For-Next o dimenticare di chiudere una parentesi. Prima ancora di poter eseguire la routine, è necessario correggere questi errori. Un bug del programma è molto più subdolo. La routine funzionerà, ma non nel modo previsto.

Identificare i bug

Prima di poter eseguire il debug, è necessario determinare se esiste effettivamente un bug. Potete dire che la macro contiene un bug se non funziona come dovrebbe (accidenti, questo libro è pieno di perle di saggezza, vero?). Di solito, ma non sempre, potete capirlo con una certa facilità.

Un bug, spesso (ma non sempre), diventa evidente quando Excel visualizza un messaggio d'errore runtime. La Figura 13.1 mostra un esempio. Notate che questo messaggio d'errore include un pulsante con l'etichetta Debug. Troverete ulteriori informazioni su questo argomento più avanti nel paragrafo "Il debugger".

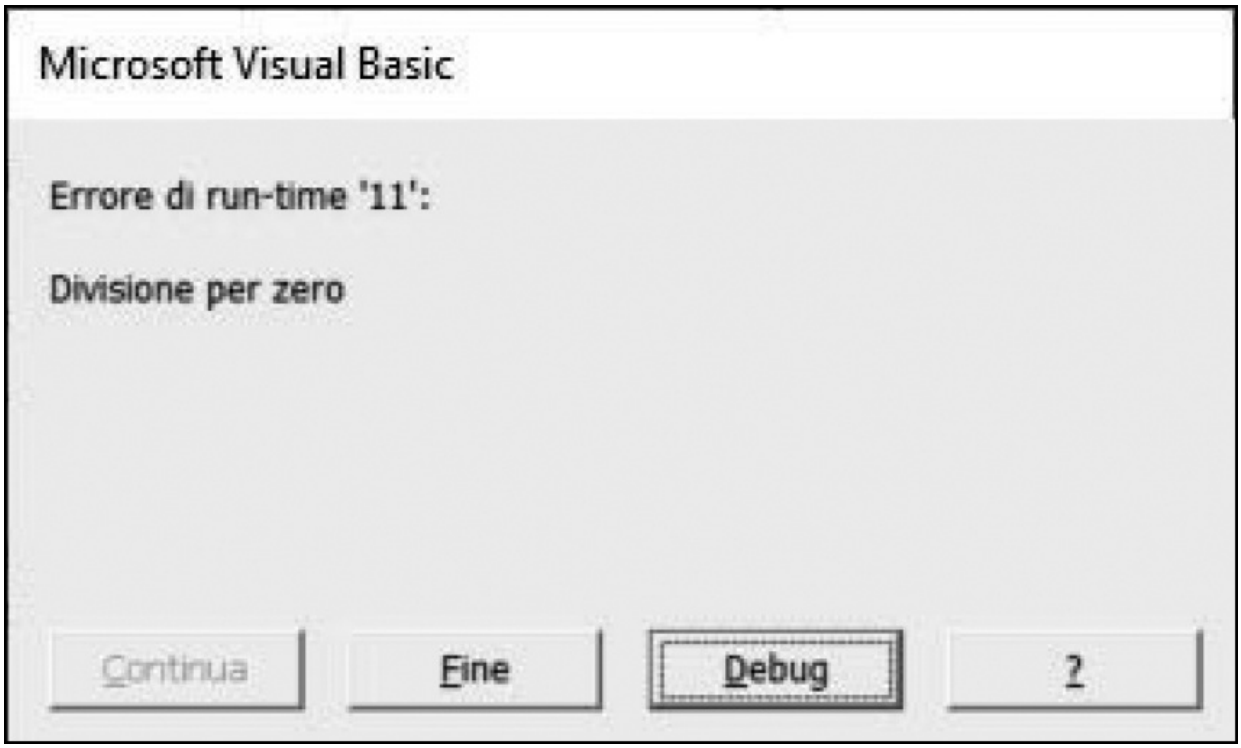

FIGURA 13.1 Un messaggio d'errore come questo spesso significa che il codice VBA contiene un bug.

Un fatto chiave noto a tutti i programmatori è che spesso i bug compaiono quando meno ve lo aspettate. Per esempio, solo perché la macro funziona bene con determinati dati, questo non significa che possiate presumere che funzioni ugualmente bene con tutti i dati.

Il miglior approccio al debug è iniziare con test approfonditi, in una varietà di condizioni di vita reale. E poiché non potete annullare le modifiche apportate alla cartella di lavoro dal codice VBA, è sempre consigliabile utilizzare una copia di backup della cartella di lavoro, da dedicare ai test.

Tecniche di debugging

In questa parte, esploriamo i quattro metodi più comuni per il debug del codice VBA di Excel:

- Esaminare bene il codice.
- Inserire funzioni MsgBox in specifici punti del codice.
- Inserire istruzioni Debug.Print.
- Usare gli strumenti di debug integrati di Excel.

Esaminare bene il codice

Forse la tecnica di debug più semplice consiste semplicemente nel dare un'occhiata al codice per vedere se si riesce a trovare il problema. Questo metodo, ovviamente, richiede conoscenza ed esperienza. In altre parole, dovete sapere che cosa state facendo. Se siete fortunati, l'errore salta fuori e vi date una manata sulla fronte dicendo a voi stessi: "Eccolo!". Ma spesso, scoprirete gli errori magari dopo aver lavorato al programma per otto ore di seguito, alle 2 del mattino, in piedi solo a caffeina e forza di volontà. In momenti come questo, siete già fortunati se riuscite anche a leggere davvero il codice; figuriamoci trovare i bug. Quindi, non sorprendetevi se esaminare semplicemente il codice

non sarà sufficiente a farvi trovare e cancellare tutti i bug che contiene.

Usare la funzione MsgBox

Un problema comune di molti programmi è che una o più variabili non assumono i valori che ci si aspetta. In questi casi, il monitoraggio della/e variabile/i nell'esecuzione del codice è una tecnica utile per il debug. Un modo per farlo è inserire nella routine delle funzioni MsgBox temporanee. Se disponete, per esempio, di una variabile denominata CellCount, potete inserire la seguente istruzione:

```
MsgBox CellCount
```

Quando eseguirete la routine, la funzione MsgBox visualizzerà il valore contenuto in CellCount.

Spesso è utile visualizzare i valori di due o più variabili nella finestra del messaggio. L'istruzione seguente mostra il valore corrente di due variabili: LoopIndex (1) e CellCount (72), separati da uno spazio.

```
MsgBox LoopIndex & " " & CellCount
```

Notate che le due variabili sono combinate con l'operatore di concatenamento (&) e con un carattere di spazio. Altrimenti, la finestra MsgBox unirebbe i due valori, facendoli apparire come un singolo valore. Inoltre potreste utilizzare la costante predefinita, vbNewLine, al posto del carattere spazio. vbNewLine inserisce un'interruzione di riga, che visualizza il testo successivo su una nuova riga. L'istruzione seguente mostra tre variabili, ciascuna su una riga distinta (vedi la Figura 13.2):

```
MsgBox LoopIndex & vbNewLine & CellCount & vbNewLine & MyVal
```

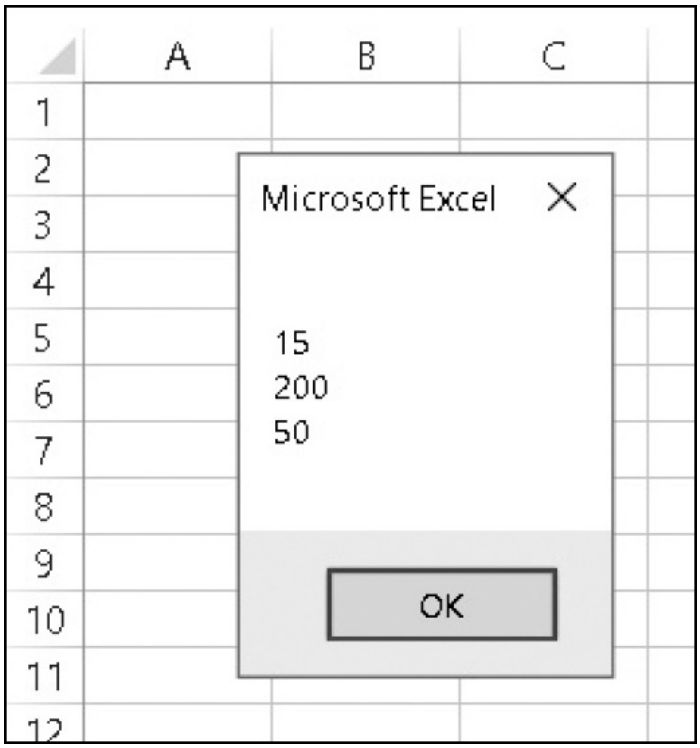

FIGURA 13.2 Uso di una finestra per visualizzare il valore di tre variabili.

Questa tecnica non si limita al monitoraggio delle variabili. Mentre il codice è in esecuzione potete utilizzare una finestra MsgBox per visualizzare tutti i tipi di informazioni utili. Se il codice, per esempio, esamina una serie di fogli, la seguente istruzione visualizza il nome e il tipo del foglio attivo:

```
MsgBox ActiveSheet.Name & " " & TypeName(ActiveSheet)
```

Se la finestra mostra qualcosa di inaspettato, premete Ctrl + Interr e una finestra di dialogo che vi dirà che l'esecuzione del codice è stata interrotta; come illustrato nella Figura 13.3, avete quattro scelte:

- fare clic sul pulsante Continua e il codice continuerà l'esecuzione;
- fare clic sul pulsante Fine e l'esecuzione si fermerà;
- fare clic sul pulsante Debug e il VBE entrerà in modalità di Debug (trattata più avanti nel paragrafo "Il debugger");
- fare clic sul pulsante della Guida (?) e una schermata vi dirà che avete premuto Ctrl + Interr… molto utile, vero?

CONSIGLIO

Se la tastiera non ha un tasto Interr, provate a premere Ctrl + Bloc Scorr.

RICORDA!

Siete liberi di usare quando e quanto volete le funzioni MsgBox quando eseguite il debug del codice. Assicuratevi solo di rimuoverle tutte dopo aver identificato e risolto il problema.

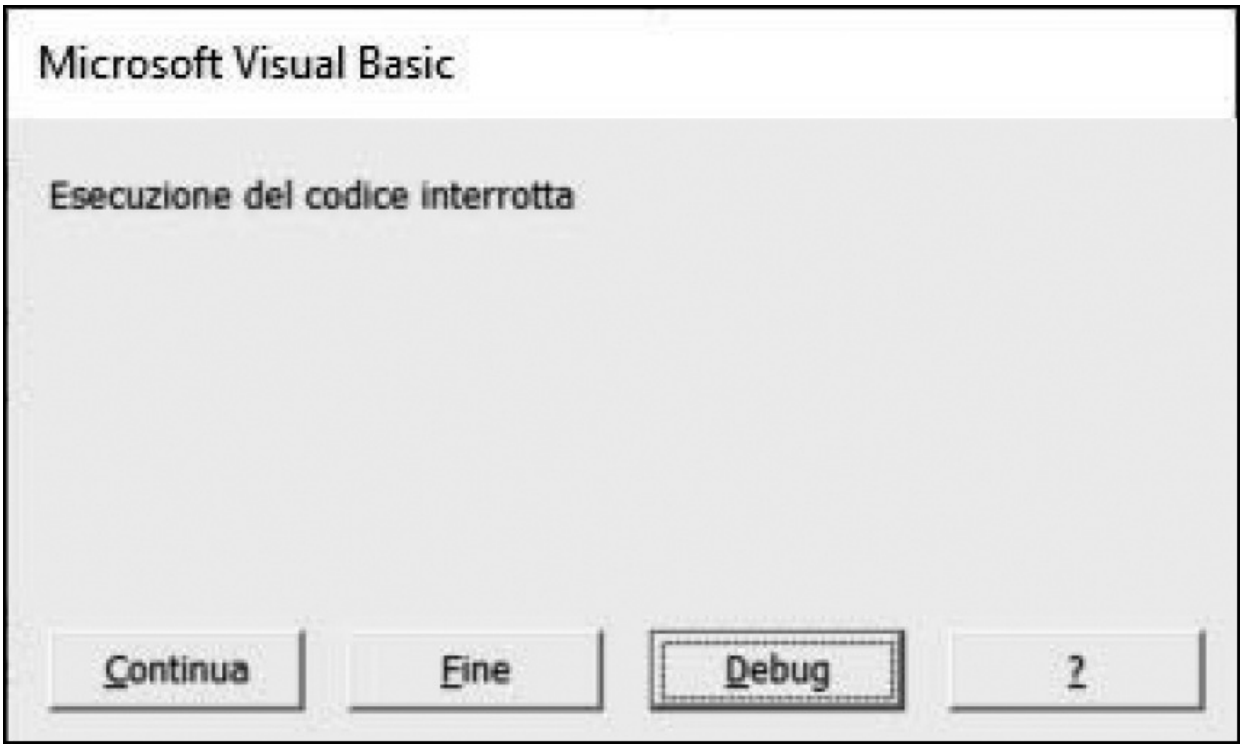

FIGURA 13.3 Premendo Ctrl + Interr interrompete l'esecuzione del codice e otterrete alcune scelte.

Inserire istruzioni Debug.Print

In alternativa all'utilizzo nel codice delle funzioni MsgBox, potete inserire una o più istruzioni temporanee Debug.Print. Potete usare queste istruzioni per stampare il valore di una o più variabili nella finestra Immediata. Ecco un esempio che mostra i valori di tre variabili:

```
Debug.Print LoopIndex, CellCount, MyVal
```

Notate che le variabili sono separate da virgole. Con una singola istruzione Debug. Print potete visualizzare tutte le variabili che desiderate.

RICORDA!

Debug.Print invia l'output alla finestra Immediata anche se la finestra è nascosta. Se la finestra Immediata di VBE non è visibile, premete Ctrl + G (o scegliete Visualizza ➪ Finestra Immediata). La Figura 13.4 mostra alcuni risultati nella finestra Immediata.

A differenza di MsgBox, le istruzioni Debug.Print non fermano l'esecuzione del codice. Quindi è necessario tenere d'occhio la finestra Immediata per vedere cosa sta succedendo.

Dopo aver eseguito il debug del codice, assicuratevi di rimuovere o di trasformare in commenti tutte le istruzioni

Debug.Print. Anche le grandi aziende come Microsoft di tanto in tanto dimenticano di rimuovere le loro istruzioni Debug.Print. In diverse versioni precedenti di Excel, ogni volta che veniva aperto il componente aggiuntivo Analysis ToolPak, potevano essere visualizzati strani messaggi nella finestra Immediata. Questo problema è stato finalmente risolto a partire da Excel 2007.

FIGURA 13.4 Un'istruzione Debug.Print invia l'output alla finestra Immediata.

Usare il debugger VBA

I progettisti di Excel hanno familiarità con il concetto di bug. Di conseguenza, Excel include tutta una serie di strumenti di debug che possono aiutare a correggere i problemi presenti nel codice VBA. Il debugger VBA è l'argomento del prossimo paragrafo.

Il debugger

Questa parte del capitolo illustra i dettagli dell'utilizzo degli strumenti di debugging di Excel. Questi strumenti sono molto più potenti delle rudimentali tecniche trattate nel paragrafo precedente. Ma insieme al potere viene sempre la responsabilità: l'utilizzo degli strumenti di debug richiede del lavoro di predisposizione.

Impostare i breakpoint nel codice

Potete certamente disseminare nel codice tante funzioni MsgBox per monitorare i valori di determinate variabili (come descritto in precedenza nel paragrafo "Usare la funzione MsgBox"). La visualizzazione di una finestra MsgBox interrompe l'esecuzione del codice; facendo clic sul pulsante OK, l'esecuzione riprende.

Non sarebbe bello se si potesse sospendere l'esecuzione di una routine, dare un'occhiata al valore *di una* delle variabili e poi continuare l'esecuzione? Bene, questo è esattamente quello che potete fare impostando un *breakpoint*, in italiano un "punto d'arresto". Potete impostare un breakpoint nel codice VBA in diversi modi.

- » Portate il cursore sull'istruzione in cui volete sospendere l'esecuzione; poi premete F9.
- » Fate clic sul margine grigio a sinistra dell'istruzione in cui desiderate interrompere l'esecuzione.
- » Posizionate il cursore sull'istruzione in cui volete sospendere l'esecuzione; poi selezionate Debug ➪ Imposta/rimuovi punto di interruzione.
- » Fate clic destro su un'istruzione e selezionate dal menu contestuale il comando Imposta/rimuovi ➪ Punto di interruzione.

I risultati dell'impostazione di un punto di interruzione sono illustrati nella Figura 13.5. Excel evidenzia la riga per ricordarvi che in quel punto avete impostato un *breakpoint*; inserisce anche un grosso punto sul margine.

FIGURA 13.5 L'istruzione evidenziata segna un punto di interruzione in questa routine.

Quando si esegue la routine, Excel passa alla modalità *Interruzione* prima che venga eseguita la riga con il punto di interruzione. Nella modalità Interruzione, nella barra del titolo di VBE viene visualizzata la parola `[interruzione]`. Per uscire dalla modalità Interruzione e continuare l'esecuzione, premete F5 o fate clic sul pulsante Esegui Sub/UserForm nella barra degli strumenti di VBE. Per ulteriori informazioni, consultate il paragrafo "Usare la finestra Espressioni di controllo" più avanti in questo stesso capitolo.

CONSIGLIO

Per rimuovere rapidamente un breakpoint, fate clic sul grosso punto nel margine grigio o portate il cursore sulla riga evidenziata e premete F9. Per rimuovere tutti i punti di interruzione presenti nel modulo, premete la combinazione di tasti Ctrl + Maiusc + F9.

VBA ha anche una parola riservata, che potete inserire come istruzione, e che forza la modalità Interruzione:

```
Stop
```

Quando il codice raggiunge la parola riservata Stop, VBA entra in modalità Interruzione.

Cos'è la modalità Interruzione? Potete considerarla come uno stato di animazione sospesa. Il codice VBA smette di funzionare e l'istruzione corrente viene evidenziata in giallo brillante. Nella modalità Interruzione, potete fare le seguenti cose.

- Digitare istruzioni VBA nella finestra Immediata (per ulteriori dettagli consultate il prossimo paragrafo).
- Premere F8 per scorrere il codice una riga alla volta, in modo da controllarlo mantenendo il programma in pausa.
- Spostare il puntatore del mouse su una variabile per visualizzarne il valore in una piccola finestra a scomparsa.
- Saltare a un'istruzione successiva e continuare l'esecuzione da quel punto (o tornare indietro di qualche istruzione).
- Modificare un'istruzione e poi continuare.

CONSIGLIO

La Figura 13.6 mostra alcune azioni di debug. Viene impostato un breakpoint (notate il grosso punto) e il tasto F8

viene poi utilizzato per scorrere il codice riga per riga (notate la freccia che punta all'istruzione corrente).

FIGURA 13.6 Una situazione tipica, in modalità Interruzione.

Usare la finestra Immediata

La finestra Immediata potrebbe non essere visibile in VBE. Potete visualizzarla in qualsiasi momento premendo Ctrl + G.

In modalità Interruzione, la finestra Immediata è particolarmente utile per trovare il valore attuale di qualsiasi variabile. Per esempio, se volete conoscere il valore attuale della variabile denominata CellCount, immettete nella finestra Immediata quanto segue e premete Invio:

```
Print CellCount
```

Potete risparmiare alcuni millisecondi usando un punto interrogativo al posto della parola riservata `Print`, in questo modo:

```
? CellCount
```

La finestra Immediata vi consente di fare altre cose oltre a controllare i valori delle variabili. Per esempio, potete modificare il valore di una variabile, attivare un altro foglio e perfino aprire una nuova cartella di lavoro. Assicuratevi che il comando che inserite sia un'istruzione VBA valida.

CONSIGLIO

Potete utilizzare la finestra Immediata anche quando Excel non è in modalità Interruzione. La finestra Immediata è un luogo comodo per provare piccoli frammenti di codice prima di incorporarli nelle routine.

Attraversare il codice, passo dopo passo

In modalità Interruzione, potete anche scorrere il codice riga per riga. Ogni volta che premete F8 viene eseguita un'istruzione. Durante l'esecuzione riga per riga del codice, potete attivare in qualsiasi momento la finestra Immediata, per verificare lo stato delle variabili.

CONSIGLIO

Potete utilizzare il mouse per decidere quale istruzione VBA deve essere eseguita successivamente. Se posizionate il puntatore del mouse sul margine a sinistra dell'istruzione attualmente evidenziata (che di solito è gialla), il puntatore si trasforma in una freccia rivolta verso destra. Basta trascinare il mouse sull'istruzione che desiderate eseguire e vedrete quell'istruzione diventare gialla.

Usare la finestra Espressioni di controllo

In alcuni casi, potreste voler sapere se una determinata variabile o espressione assume un particolare valore. Supponiamo che una routine lavori ciclicamente su 1.000 celle. Notate che alla novecentesima iterazione del ciclo si verifica un problema. Bene, potreste anche inserire un breakpoint nel ciclo, ma ciò vi obbligherebbe a rispondere a 899 richieste prima che il codice raggiunga finalmente l'iterazione che desiderate controllare: non è né simpatico e neanche veloce. Una soluzione più efficiente comporta l'impostazione di un'*espressione di controllo*.

Per esempio, potete creare un'espressione di controllo che pone la routine in modalità Interruzione ogni volta che una determinata variabile assume un determinato valore, per esempio Counter = 900. Per creare un'espressione di controllo, selezionate Debug ➪ Aggiungi espressione di controllo per visualizzare la finestra di dialogo omonima (vedi la Figura 13.7).

FIGURA 13.7 La finestra di dialogo Aggiungi espressione di controllo consente di specificare una condizione che causa un'interruzione.

La finestra di dialogo Aggiungi espressione di controllo è composta da tre parti:

- **Espressione:** inserite qui un'espressione VBA o una variabile valida, per esempio Counter = 900 o anche solo Counter.
- **Contesto:** seleziona la routine e il modulo che desiderate controllare. Notate che potete selezionare anche le opzioni (tutte le routine) e (tutti i moduli).
- **Tipo di espressione di controllo:** seleziona il tipo di controllo, tramite un pulsante di opzione. La vostra scelta dipende dall'espressione che avete inserito. La prima scelta, Espressione di controllo, non provoca un'interruzione; mostra semplicemente il valore dell'espressione quando si verifica un'interruzione.

Eseguite la routine dopo aver impostato l'espressione di controllo. Le cose funzionano normalmente fino a che non è soddisfatta l'espressione di controllo (in base al tipo di controllo specificato). Quando ciò accade, Excel entra in modalità Interruzione (avete impostato il Tipo di espressione di controllo su Interrompi se il valore è vero, giusto?). Da lì, potete scorrere passo-passo il codice o utilizzare la finestra Immediata per eseguirne il debug.

Quando create un controllo, VBE visualizza la finestra Espressioni di controllo, rappresentata nella Figura 13.8. Questa finestra mostra il valore di tutte le espressioni di controllo che avete definito. In questa figura, la variabile Counter ha raggiunto il valore 900, il che ha fatto in modo che Excel entrasse in modalità Interruzione.

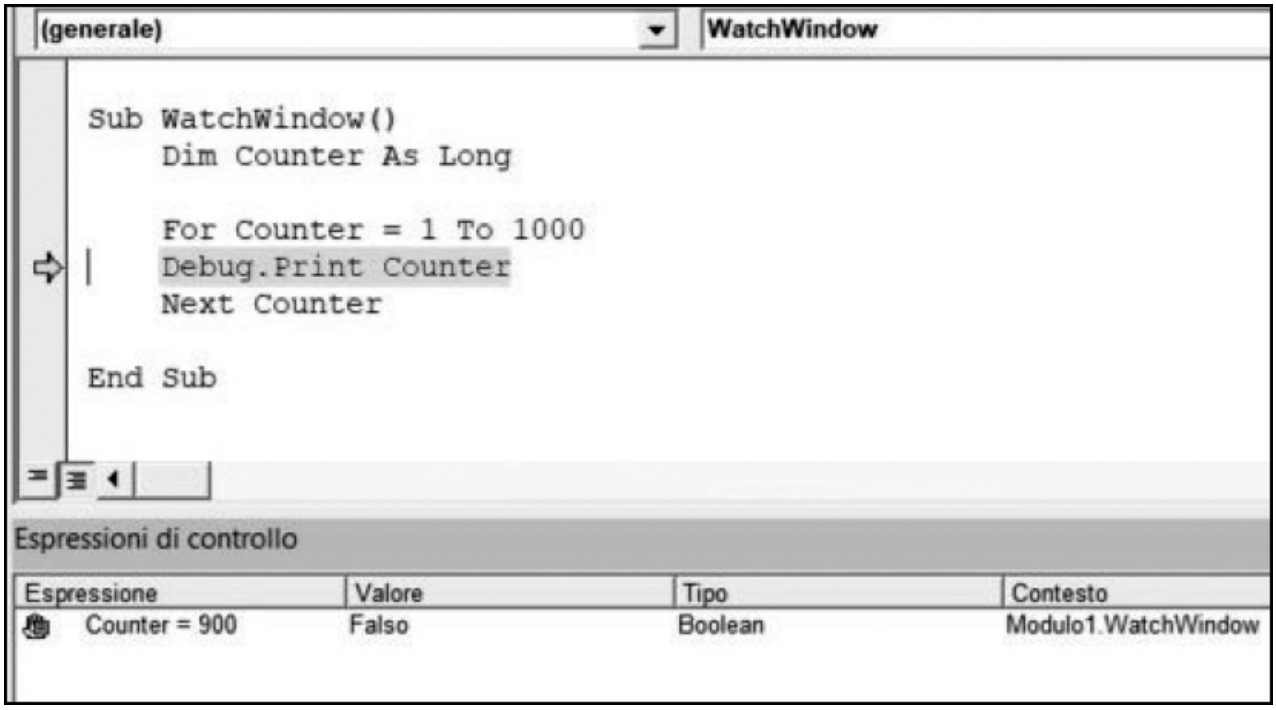

FIGURA 13.8 La finestra Espressioni di controllo mostra tutte le espressioni di controllo che avete impostato.

Per rimuovere un'espressione di controllo, fate clic destro nel riquadro Espressioni di controllo e scegliete dal menu contestuale il comando Elimina espressione di controllo.

Il modo migliore per capire come funziona questa istruzione è usarla e provare varie opzioni. In poco tempo, probabilmente vi chiederete come avete fatto, finora, a farne a meno.

Usare la finestra Variabili locali

Un altro utile aiuto per il debug è la finestra Variabili locali. Potete visualizzarla scegliendo in VBE Visualizza ➪ Finestra Variabili locali. In modalità Interruzione, questa finestra mostra un elenco di tutte le variabili che sono locali alla routine corrente (vedi la Figura 13.9). La cosa bella di questa finestra è che vi evita di aggiungere manualmente molte espressioni di controllo per vedere il contenuto delle variabili. Il VBE farà tutto il lavoro per voi.

FIGURA 13.9 La finestra Variabili locali visualizza tutte le variabili locali e il loro contenuto.

Suggerimenti per la riduzione dei bug

Non esiste un modo sicuro per eliminare completamente i bug nei programmi VBA. Ecco alcuni suggerimenti per aiutarvi a ridurli al minimo.

- **Usate un'istruzione Option Explicit all'inizio dei vostri moduli.** Questa istruzione vi obbliga a definire il tipo di dati per ogni variabile che utilizzate. Dovrete fare più lavoro, ma eviterete l'errore, molto comune, di digitare erroneamente un nome di variabile. E ha un vantaggio collaterale: le vostre routine saranno un po' più veloci.
- **Formattate il codice con l'indentazione.** I rientri dell'indentazione aiutano a delineare i diversi segmenti di codice. Per esempio, se il vostro programma contiene diversi cicli For-Next annidati, un'indentazione coerente vi aiuterà a seguirli meglio.
- **Fate attenzione con l'istruzione On Error Resume Next.** Questa istruzione fa sì che Excel ignori gli errori e continui a eseguire la routine. In alcuni casi, questa istruzione fa sì che Excel ignori errori che non

dovrebbe ignorare. Il codice potrebbe contenere bug e non ve ne accorgereste.

- **Usate i commenti.** Niente è più frustrante che rivedere del codice scritto magari solo sei mesi prima e non avere la minima idea di come funzioni. Aggiungendo alcuni commenti per descriverne il funzionamento, risparmierete sicuramente molto tempo.
- **Mantenete semplici le routine Sub e Function.** Scrivendo il codice in piccoli moduli, ognuno dei quali ha un unico scopo ben definito, semplificherete il processo di debug.
- **Utilizzate il registratore di macro per identificare le proprietà e i metodi.** Quando non riuscite a ricordare il nome o la sintassi di una proprietà o di un metodo, registrate semplicemente una macro e osservate il codice che essa produce.
- **Imparate a usare il debugger di Excel.** Sebbene all'inizio possa essere un po' scoraggiante, il debugger di Excel è uno strumento utile. Investite un po' di tempo a studiarlo.

Capitolo 14

Esempi di codice VBA

IN QUESTO CAPITOLO

- **Lavorare con intervalli nel codice VBA**
- **Modificare le impostazioni booleane e non booleane**
- **Manipolare i grafici con VBA**
- **Rendere il codice VBA più veloce possibile**

Spesso un buon esempio comunica un concetto molto meglio di una lunga descrizione teorica. Con questo spirito, questo capitolo presenta diversi esempi che descrivono l'uso di varie tecniche di programmazione VBA.

Gli esempi sono organizzati nelle seguenti categorie.

- Gli intervalli.
- Le impostazioni di Excel.
- I grafici.
- Ottimizzazione del codice VBA.

Sebbene sia possibile utilizzare direttamente alcuni esempi, nella maggior parte dei casi dovrete adattarli alle vostre esigenze.

Lavorare con gli intervalli

La maggior parte dell'attività di programmazione in VBA prevede l'uso di intervalli di celle dei fogli di lavoro: Range. Per informazioni sugli oggetti Range, che controllano gli intervalli, consultate il Capitolo 8. Quando lavorate con oggetti Range, tenete presente i seguenti punti.

- Una routine VBA non ha bisogno di selezionare un intervallo, per lavorarci.
- Se il vostro codice deve selezionare un intervallo, il foglio di lavoro in cui si trova deve essere attivo.
- Il registratore di macro, in genere non produce codice efficiente. Spesso, è opportuno creare una macro usando il registratore, ma poi modificarla per renderla più efficiente.
- Spesso è una buona idea utilizzare nel codice VBA intervalli con nome. Per esempio, è meglio scrivere Range("Totale") che Range("D45"). In quest'ultimo caso, se aggiungerete una riga sopra la riga 45, dovrete modificare la macro in modo che utilizzi l'indirizzo di cella aggiornato (D46). Per assegnare un nome a un intervallo di celle basta usare il comando Formule ➪ Nomi definiti ➪ Definisci nome.
- Quando si esegue una macro che funziona sull'intervallo di celle attualmente selezionato, l'utente potrebbe aver scelto intere righe o colonne di dati. Nella maggior parte dei casi, non volete eseguire il vostro ciclo su ogni singola cella della selezione; sarebbe solo uno spreco di tempo. La vostra macro dovrebbe creare innanzitutto un sottoinsieme della selezione, costituito solo dalle celle non vuote.
- Excel consente di definire selezioni multiple. Per esempio, potete selezionare un intervallo, premere Ctrl e selezionare un altro intervallo con il mouse. Il codice può verificare se ha a che fare con una selezione multipla e intraprendere azioni appropriate.

CONSIGLIO

Gli esempi di questa parte, disponibili (in originale inglese) sul sito web di questo libro, illustrano questi concetti.

RICORDA!

Se preferite digitare questi esempi, premete Alt + F11 per attivare il VBE. Quindi inserite un modulo VBA e digitate il codice proposto. Assicuratevi che la cartella di lavoro sia impostata correttamente. Per esempio, se l'esempio utilizza due fogli denominati Foglio1 e Foglio2, assicuratevi che la cartella di lavoro contenga tali fogli.

Copiare un intervallo

La copia di un intervallo è tra le attività di Excel preferite di tutti i tempi. Quando si accende il registratore di macro e si copia un intervallo da A1:A5 a B1:B5, si ottiene questa macro VBA:

```
Sub CopyRange()
     Range("A1:A5").Select
     Selection.Copy
     Range("B1").Select
     ActiveSheet.Paste
     Application.CutCopyMode = False
End Sub
```

Notate l'ultima istruzione. È stata generata premendo Esc, per togliere di mezzo la cornice tratteggiata che appare nel foglio di lavoro quando si copia un intervallo.

Questa macro funziona correttamente, ma potete copiare un intervallo in modo molto più efficiente. Potete produrre lo stesso risultato con la seguente macro di una sola riga, che non seleziona alcuna cella (e inoltre non richiede l'impostazione di CutCopyMode su False):

```
Sub CopyRange2()
     Range("A1:A5").Copy Range("B1")
End Sub
```

Questa routine sfrutta il fatto che il metodo Copy può utilizzare un argomento che specifica la destinazione.

In alternativa, avete la possibilità di specificare l'argomento Destination, per migliorare la leggibilità del codice. Per esempio:

```
Sub CopyRange2()
     Range("A1:A5").Copy Destination:=Range("B1")
End Sub
```

Copiare un intervallo di dimensioni variabili

In molti casi, è necessario copiare un intervallo di celle, ma non se ne conoscono le dimensioni esatte in termini di righe e colonne. Per esempio, potreste avere una cartella di lavoro che registra le vendite settimanali. Il numero di righe cambierà ogni volta che aggiungerete nuovi dati.

La Figura 14.1 mostra un intervallo in un foglio di lavoro. Questo intervallo comprende più righe e il numero di righe cambia da un giorno al successivo. Poiché non si conosce l'intervallo esatto in un dato momento, è necessario un modo per scrivere del codice che non utilizzi un intervallo ben specifico.

	A	B	C	D
1	Data	Chiamate	Ordini	
2	05/04/2019	452	89	
3	06/04/2019	546	102	
4	07/04/2019	587	132	
5	08/04/2019	443	65	
6	09/04/2019	609	156	
7	10/04/2019	592	92	
8	11/04/2019	487	95	
9	12/04/2019	601	105	
10	13/04/2019	515	133	
11	14/04/2019	540	122	
12				
13				

FIGURA 14.1 Questo intervallo può essere costituito da un numero qualsiasi di righe.

La seguente macro mostra come copiare questo intervallo da Foglio1 a Foglio2 (iniziando dalla cella A1). Utilizza la proprietà CurrentRegion, che restituisce un oggetto Range che corrisponde al blocco di celle intorno a una determinata cella. In questo caso, la cella è A1.

```
Sub CopyCurrentRegion()
     Range("A1").CurrentRegion.Copy
     Sheets("Foglio2").Select
     Range("A1").Select
     ActiveSheet.Paste
     Sheets("Foglio1").Select
     Application.CutCopyMode = False
End Sub
```

L'uso della proprietà CurrentRegion equivale a selezionare il comando Home ➪ Mod ➪ Trova e seleziona ➪ Vai a formato speciale (che visualizza la finestra di dialogo Vai a formato speciale) e alla successiva selezione dell'opzione Area corrente. Per vedere come funziona, registrate le vostre azioni mentre richiamate quel comando. Generalmente, la CurrentRegion è costituita da un blocco rettangolare di celle, circondato da una o più righe o colonne vuote.

Potete rendere questa macro ancora più efficiente evitando di selezionare la destinazione. La seguente macro sfrutta il fatto che il metodo Copy può utilizzare un argomento per l'intervallo di destinazione:

```
Sub CopyCurrentRegion2()
     Range("A1").CurrentRegion.Copy Sheets("Foglio2").Range("A1")
End Sub
```

CONSIGLIO

Se i dati sono sotto forma di tabella, la cosa è perfino più semplice (per creare una tabella in Excel, selezionate il comando Inserisci ➪ Tabelle ➪ Tabella). La tabella ha un nome (in questo caso Tabella1) e si espande automaticamente quando le vengono aggiunti nuovi dati.

```
Sub CopyTable()
     Range("Tabella1").Copy Sheets("Foglio2").Range("A1")
End Sub
```

Se provate a usare questa routine, scoprirete che la riga di intestazione della tabella non viene copiata, perché il nome Tabella1 non include quella riga. Se è necessario includere la riga di intestazione, modificare il riferimento alla tabella in:

```
Range("Tabella1[#All]")
```

Selezionare fino alla fine una riga o una colonna

Probabilmente avete l'abitudine di usare le combinazioni di tasti Ctrl + Maiusc + freccia destra e Ctrl + Maiusc + freccia giù per selezionare un intervallo composto da tutto ciò che si trova fra la cella attiva e la fine di una riga o di una colonna. Come certo immaginate, potete scrivere una macro che esegue questi tipi di selezioni.

Potete utilizzare la proprietà CurrentRegion per selezionare un intero blocco di celle. Ma che cosa succede se desiderate selezionare, per esempio, una colonna in un blocco di celle? Fortunatamente, VBA prevede questo tipo di azione. La seguente routine VBA seleziona un intervallo che parte dalla cella attiva e si estende fino alla cella appena sopra la prima cella vuota della colonna. Dopo aver selezionato l'intervallo, potete farne quello che volete: copiarlo, spostarlo, formattarlo e così via.

```
Sub SelectDown()
     Range(ActiveCell, ActiveCell.End(xlDown)).Select
End Sub
```

Potete ottenere anche manualmente questo tipo di selezione: selezionate la prima cella, tenete premuto il tasto Maiusc, premete Fine e poi premete il tasto freccia giù.

Questo esempio utilizza il metodo End dell'oggetto ActiveCell, che restituisce un oggetto Range. Il metodo End accetta un argomento, che può essere una delle seguenti costanti:

- xlUp
- xlDown
- xlToLeft
- xlToRight

Tenete presente che non è necessario selezionare un intervallo prima di operarvi. La seguente macro applica la formattazione in grassetto a un intervallo di dimensioni variabili (a colonna singola) senza però selezionare l'intervallo:

```
Sub MakeBold()
     Range(ActiveCell, ActiveCell.End(xlDown)).Font.Bold = True
End Sub
```

Selezionare una riga o una colonna

La seguente routine mostra come selezionare la colonna che contiene la cella attiva. Utilizza la proprietà EntireColumn, che restituisce un oggetto Range che è costituito da una colonna, intera:

```
Sub SelectColumn()
     ActiveCell.EntireColumn.Select
End Sub
```

Come sicuramente immaginate, VBA offre anche una proprietà EntireRow, che restituisce un oggetto Range costituito da un'intera riga.

Spostare un intervallo

Si sposta un intervallo tagliandolo negli Appunti e poi incollandolo in un'altra area. Se registrate le azioni mentre eseguite un'operazione di spostamento, il registratore di macro genera del codice simile al seguente:

```
Sub MoveRange()
     Range("A1:C6").Select
     Selection.Cut
     Range("A10").Select
     ActiveSheet.Paste
End Sub
```

Come nel caso dell'esempio di copia, sempre in questo capitolo, questo non è il modo più efficace per spostare un intervallo di celle. In effetti, potete spostare un intervallo con una singola istruzione VBA:

```
Sub MoveRange2()
     Range("A1:C6").Cut Range("A10")
End Sub
```

Questa macro sfrutta il fatto che il metodo Cut può utilizzare un argomento, che specifica la destinazione. Notate inoltre che l'intervallo non è stato selezionato. Il cursore della cella rimane nella sua posizione originale.

Eseguire un ciclo su un intervallo in modo efficiente

A molte macro capita di dover eseguire un'operazione su ogni cella di un intervallo o di eseguire determinate azioni in base al contenuto di ciascuna cella. Queste macro di solito includono un ciclo For-Next che elabora ciascuna cella

dell'intervallo.

L'esempio seguente mostra come eseguire un ciclo su un intervallo di celle. In questo caso, l'intervallo è la selezione corrente. Una variabile oggetto denominata Cell fa riferimento alla cella da elaborare. All'interno del loop For Each-Next, l'istruzione valuta la cella e, se contiene un valore positivo, applica la formattazione in grassetto.

```
Sub ProcessCells()
     Dim Cell As Range
     For Each Cell In Selection
         If Cell.Value > 0 Then Cell.Font.Bold = True
     Next Cell
End Sub
```

Questo esempio funziona, ma che cosa succede se la selezione dell'utente è un'intera colonna o riga? Questa non è un'eventualità rara, perché Excel consente di eseguire operazioni su intere righe o colonne. In tal caso, la macro impiegherà un'eternità, perché dovrà analizzare tutte le celle (ben 1.048.576) della colonna, comprese le celle vuote. Per rendere più efficiente la macro, è necessario un modo per farle elaborare solo le celle non vuote.

La seguente routine fa proprio questo, usando il metodo SpecialCells (fate riferimento alla Guida VBA per i dettagli sui suoi argomenti). Questa routine utilizza la parola riservata Set per creare due nuovi oggetti Range: il sottoinsieme della selezione costituito da celle contenenti valori costanti e il sottoinsieme della selezione costituito dalle celle contenenti formule. La routine elabora solo questi due sottoinsiemi, con l'effetto netto di saltare tutte le celle vuote. Carino, vero?

```
Sub SkipBlanks()
     Dim ConstantCells As Range
     Dim FormulaCells As Range
     Dim cell As Range
'   Ignora gli errori
     On Error Resume Next
'   Elabora le costanti
     Set ConstantCells = Selection.SpecialCells(xlConstants)
     For Each cell In ConstantCells
         If cell.Value > 0 Then
              cell.Font.Bold = True
         End If
     Next cell

'   Elabora le formule
     Set FormulaCells = Selection.SpecialCells(xlFormulas)
     For Each cell In FormulaCells
         If cell.Value > 0 Then
              cell.Font.Bold = True
         End If
     Next cell
End Sub
```

La routine SkipBlanks funziona in modo sempre veloce, indipendentemente da ciò che selezionate. Per esempio, potete selezionare un intervallo, tutte le colonne dell'intervallo, tutte le righe dell'intervallo e perfino l'intero foglio di lavoro. È un enorme miglioramento rispetto alla routine ProcessCells presentata in precedenza.

Notate l'uso della seguente istruzione in questo codice:

```
On Error Resume Next
```

Questa istruzione chiede a Excel di ignorare eventuali errori che si possono verificare e di elaborare semplicemente la successiva istruzione (per una descrizione della gestione degli errori consultate il Capitolo 12). Questa istruzione è necessaria perché, se nessuna cella è idonea, il metodo SpecialCells genera un errore.

Usare il metodo SpecialCells equivale a selezionare il comando Home ➪ Modifica ➪ Trova e seleziona ➪ Vai a formato speciale e poi selezionare l'opzione Costanti o l'opzione Formule. Per avere un'idea del suo funzionamento, registrate le azioni mentre eseguite quel comando e selezionate varie opzioni.

Eseguire un ciclo su un intervallo in modo efficiente (Parte II)

E ora, il *sequel*. Questo paragrafo presenta un altro modo per elaborare le celle in modo efficiente. Questo metodo sfrutta la proprietà UsedRange, che restituisce un oggetto Range formato solo dall'area utilizzata del foglio di lavoro. Utilizza anche il metodo Intersect, che restituisce un oggetto Range formato dalle celle in comune fra due intervalli.

Ecco una variante della routine SkipBlanks del paragrafo precedente:

```
Sub SkipBlanks2()
     Dim WorkRange As Range
     Dim cell As Range
     Set WorkRange = Intersect(Selection, ActiveSheet.UsedRange)
     For Each cell In WorkRange
         If cell.Value > 0 Then
              cell.Font.Bold = True
         End If
     Next cell
```

```
End Sub
```

La variabile oggetto WorkRange è composta dalle celle in comune fra la selezione dell'utente e l'intervallo utilizzato del foglio di lavoro. Pertanto, se l'utente seleziona un'intera colonna, WorkRange conterrà solo le celle di quella colonna che sono anche nell'area utilizzata del foglio di lavoro. Il tutto in modo veloce ed efficiente, senza perdite di tempo nell'elaborazione di celle che si trovano all'esterno dell'area utilizzata del foglio di lavoro.

Richiedere un valore per una cella

Come potete vedere nella Figura 14.2, la funzione InputBox permette di ottenere un valore dall'utente. Poi potrete inserire tale valore in una cella. La seguente routine mostra come chiedere all'utente un valore e poi collocare il valore ottenuto nella cella A1 del foglio di lavoro attivo, utilizzando una sola istruzione:

```
Sub GetValue()
     Range("A1").Value = InputBox(_
         "Inserire il valore per la cella A1")
End Sub
```

FIGURA 14.2 Uso della funzione VBA InputBox per ottenere un valore dall'utente.

Se provate questo esempio, scoprirete che facendo clic sul pulsante Annulla nella finestra si cancella il valore attualmente presente nella cella A1. Cancellare i dati dell'utente non è mai una buona cosa, in programmazione. Facendo clic su Annulla, l'utente non dovrebbe eseguire nessuna azione.

La seguente macro mostra un approccio migliore: utilizza una variabile (x) per memorizzare il valore inserito dall'utente. Se il valore non è vuoto (ovvero, l'utente non ha fatto clic su Annulla), il valore di x viene inserito nella cella A1. Altrimenti, non succede nulla.

```
Sub GetValue2()
     Dim x as Variant
     x = InputBox("Inserire il valore per la cella A1")
     If x <> "" Then Range("A1").Value = x
End Sub
```

La variabile x è definita di tipo Variant, perché potrebbe essere un numero o anche una stringa vuota (se l'utente fa clic su Annulla).

Determinare il tipo della selezione

Se progettate una macro che deve lavorare con un intervallo selezionato, dovrà essere in grado di determinare se un intervallo è stato effettivamente selezionato. Se viene selezionato qualcosa di diverso da un intervallo (un grafico, una forma), la macro probabilmente andrà in errore. La seguente istruzione utilizza la funzione TypeName per visualizzare il tipo di oggetto attualmente selezionato:

```
MsgBox TypeName(Selection)
```

Se viene selezionato un oggetto Range, MsgBox visualizza Range. Se la macro funziona solo con gli intervalli, potete utilizzare un'istruzione If per assicurarvi che sia selezionato un intervallo. Questo esempio visualizza un messaggio ed esce dalla routine se la selezione corrente non è un oggetto Range:

```
Sub CheckSelection()
     If TypeName(Selection) <> "Range" Then
         MsgBox "Selezionare un intervallo."
```

```
        Exit Sub
    End If
'   ... [Altre istruzioni]
End Sub
```

Identificare una selezione multipla

Come sapete, Excel consente di creare selezioni multiple, premendo Ctrl mentre si scelgono gli oggetti o gli intervalli. Con alcune macro ciò può causare problemi. Per esempio, non è possibile copiare una selezione multipla composta da celle non adiacenti. Se tentaste di farlo, Excel vi rimproverebbe con il messaggio rappresentato nella Figura 14.3.

FIGURA 14.3 Excel non gradisce le operazioni di Copia di celle non adiacenti.

La seguente macro mostra come determinare se l'utente ha effettuato una selezione multipla, in modo che la macro possa intraprendere l'azione appropriata:

```
Sub MultipleSelection()
    If Selection.Areas.Count > 1 Then
        MsgBox "Non usare selezioni multiple."
        Exit Sub
    End If
'   … [Altre istruzioni]
End Sub
```

Questo esempio utilizza il metodo Areas, che restituisce una collezione di tutti gli intervalli presenti nella selezione. La proprietà Count restituisce il numero di oggetti presenti nella collezione.

Modificare le impostazioni di Excel

Alcune delle macro più utili sono semplici routine che modificano una o più impostazioni di Excel. Per esempio, se usate frequentemente la finestra di dialogo Opzioni di Excel per modificare un'impostazione, questo intervento è un buon candidato per creare una semplice macro che vi farà risparmiare tempo.

Questo paragrafo presenta due esempi che mostrano come modificare le impostazioni di Excel. Potete applicare i principi generali presentati da questi esempi anche ad altre operazioni che modificano le impostazioni.

Modificare le impostazioni booleane

Come un interruttore della luce, un'impostazione *booleana* può essere attivata o disattivata. Per esempio, potreste voler creare una macro che attiva e disattiva la visualizzazione delle interruzioni di pagina nel foglio di lavoro. Dopo aver stampato o visualizzato l'anteprima di un foglio di lavoro, Excel visualizza delle linee tratteggiate per indicare le interruzioni di pagina. Molti trovano queste linee tratteggiate molto fastidiose. Sfortunatamente, l'unico modo per sbarazzarsi di loro consiste nell'aprire la finestra di dialogo Opzioni di Excel, fare clic sulla scheda Impostazioni avanzate e scorrere verso il basso fino a trovare la casella di controllo Mostra interruzioni di pagina. Se attivate il registratore di macro mentre modificate l'opzione, Excel genera il seguente codice:

```
ActiveSheet.DisplayPageBreaks = False
```

D'altra parte, se, quando registrate la macro, le interruzioni di pagina non sono visibili, Excel genera il seguente codice:

```
ActiveSheet.DisplayPageBreaks = True
```

Ciò potrebbe portare a concludere che sarebbero necessarie due macro: una per attivare la visualizzazione delle interruzioni di pagina e una per disattivarla. Non è così. La seguente routine utilizza l'operatore Not, che trasforma True in False e False in True. L'esecuzione della routine TogglePageBreaks è un modo semplice per alternare la visualizzazione dell'interruzione di pagina da True a False e da False a True:

```
Sub TogglePageBreaks()
     On Error Resume Next
     ActiveSheet.DisplayPageBreaks = Not _
         ActiveSheet.DisplayPageBreaks
End Sub
```

La prima istruzione chiede a Excel di ignorare eventuali errori. Per esempio, un foglio grafico non visualizza le interruzioni di pagina, quindi se eseguite la macro quando è attivo un foglio grafico, non verrà visualizzato un messaggio d'errore.

Potete utilizzare questa tecnica per attivare o disattivare tutte le impostazioni basate su valori booleani (True o False).

Modificare le impostazioni non booleane

Per le impostazioni non booleane potete utilizzare una struttura Select Case. Questo esempio commuta la modalità di calcolo tra manuale e automatico e visualizza un messaggio che indica la modalità corrente:

```
Sub ToggleCalcMode()
     Select Case Application.Calculation
         Case xlManual
              Application.Calculation = xlCalculationAutomatic
              MsgBox "Automatic Calculation Mode"
         Case xlAutomatic
              Application.Calculation = xlCalculationManual
              MsgBox "Manual Calculation Mode"
     End Select
End Sub
```

Potete adattare questa tecnica per modificare anche altre impostazioni non booleane.

Lavorare sui grafici

I grafici sono pieni di oggetti, quindi l'uso dei grafici con VBA può essere davvero problematico, a causa di alcune differenze tra le versioni di Excel.

Immaginate, per esempio, di dover registrare una macro in Excel 2019 mentre create un semplice grafico a colonne. Mentre state ancora registrando, decidete di eliminare le linee della griglia del grafico e modificare il titolo del grafico. La macro che avete registrato sarebbe simile a questa:

```
Sub Macro1()
     ActiveSheet.Shapes.AddChart2(201, xlColumnClustered).Select
     ActiveChart.SetSourceData Source:=Range("Foglio1!$A$1:$A$3")
     ActiveChart.SetElement (msoElementPrimaryValueGridLinesNone)
     ActiveChart.ChartTitle.Select
     ActiveChart.ChartTitle.Text = "Questo è il mio grafico"
End Sub
```

Notate l'uso del metodo AddChart2 (nella prima riga di codice). Il metodo AddChart2 è stato introdotto in Excel 2013.

Ora, se doveste registrare la stessa macro in Excel 2010, otterreste questo codice:

```
Sub Macro1()
     ActiveSheet.Shapes.AddChart.Select
     ActiveChart.ChartType = xlColumnClustered
     ActiveChart.SetSourceData Source:=Range("Foglio1!$A$1:$A$3")
     ActiveChart.Axes(xlValue).MajorGridlines.Select
     Selection.Delete
     ActiveChart.SetElement (msoElementChartTitleAboveChart)
     ActiveChart.ChartTitle.Text = "Questo è il mio grafico"
End Sub
```

Che cosa significa? Significa che la macro registrata in Excel 2013 o versioni successive non funzionerà in Excel 2010. Al contrario, la macro registrata in Excel 2010 funzionerà anche in Excel 2013 e versioni successive. In altre parole, la macro di Excel 2010 offre una compatibilità in avanti. La macro di Excel 2013 (e successive) non è invece compatibile con le versioni precedenti.

WARNING

Un tipico utente di Excel probabilmente non saprà nulla dei problemi di compatibilità delle macro per la manipolazione di grafici. Ma se condividete la vostra macro con qualcuno che usa una versione precedente di Excel, lo scoprirete rapidamente. La morale? Se fate affidamento sul registratore di macro per i grafici, assicuratevi di provare le macro con tutte le versioni di Excel in cui la macro verrà eseguita.

AddChart vs AddChart2

Ecco la sintassi ufficiale del metodo AddChart (che è compatibile con Excel 2007 e versioni successive):

```
.AddChart(Type, Left, Top, Width, Height)
```

Ecco invece la sintassi del metodo AddChart2 (che è compatibile solo con Excel 2013 e versioni successive):

```
.AddChart2 (Style, XlChartType, Left, Top, Width, Height, NewLayout)
```

Come potete vedere, il metodo AddChart2 accetta diversi argomenti aggiuntivi: argomenti che specificano lo stile, il tipo di grafico e il layout. Il metodo AddChart, al contrario, crea semplicemente un grafico vuoto. Le specifiche di tale grafico dovranno essere fornite tramite altre istruzioni.

Esaminando il codice registrato si scoprono alcuni dettagli che possono essere utili per scrivere macro per grafici. Se siete curiosi, ecco una versione artigianale della macro, che crea un grafico a partire dall'intervallo selezionato:

```
Sub CreateAChart()
     Dim ChartData As Range
     Dim ChartShape As Shape
     Dim NewChart As Chart

'   Crea le variabili oggetto
     Set ChartData = ActiveWindow.RangeSelection
     Set ChartShape = ActiveSheet.Shapes.AddChart
     Set NewChart = ChartShape.Chart
     With NewChart
         .ChartType = xlColumnClustered
         .SetSourceData Source:=Range(ChartData.Address)
         .SetElement (msoElementLegendRight)
         .SetElement (msoElementChartTitleAboveChart)
         .ChartTitle.Text = "Questo è il mio grafico"
     End With
End Sub
```

La macro è compatibile con Excel 2007 e versioni successive. Il grafico creato è un grafico a colonne affiancate, con una legenda e un titolo. Questa macro per la creazione di grafici può essere personalizzata facilmente. Un modo per farlo è registrare le vostre azioni mentre modificate il grafico e poi utilizzare il codice registrato per applicare gli interventi.

CONSIGLIO

A questo proposito, date un'occhiata al costrutto With End-With, più avanti in questo capitolo. È un modo pratico per risparmiare lavoro alla tastiera e per migliorare la leggibilità del codice.

Se volete scrivere macro VBA che manipolano i grafici, dovete conoscere alcuni dei termini utilizzati. Un *grafico incorporato* in un foglio di lavoro è un oggetto ChartObject. Potete attivare un oggetto ChartObject così come attivate un foglio. La seguente istruzione attiva il ChartObject denominato Grafico 1:

```
ActiveSheet.ChartObjects("Grafico 1").Activate
```

Dopo aver attivato il grafico, potete farvi riferimento nel codice VBA con ActiveChart. Se il grafico si trova su un suo foglio grafico, non appena attivate quel foglio grafico, diverrà attivo.

PER I PIÙ CURIOSI

Un ChartObject è anche una forma, e la cosa può essere un po' confusa. Infatti, quando il codice VBA crea un grafico, inizia aggiungendo una nuova forma. Potete anche attivare un grafico selezionando l'oggetto Shape che lo contiene:

```
ActiveSheet.Shapes("Grafico 1").Select
```

WARNING

Quando fate clic su un grafico incorporato, Excel seleziona in realtà un oggetto all'interno dell'oggetto ChartObject. Potete selezionare lo stesso ChartObject premendo Ctrl mentre fate clic sul grafico incorporato.

Cambiare tipo di grafico

Ecco un'affermazione davvero poco chiara: un oggetto ChartObject funge da contenitore per un oggetto Chart. Leggetela un po' di volte e magari alla fine la cosa potrebbe avere un senso.

Per modificare un grafico con VBA, non è necessario attivarlo. Piuttosto, il metodo Chart può restituire il grafico contenuto in ChartObject. Sempre confusi? Le due prossime routine hanno lo stesso effetto: cambiano il grafico denominato Grafico 1, trasformandolo in un grafico ad area. La prima routine attiva il grafico e poi opera sul grafico attivo. La seconda non attiva il grafico. Utilizza invece la proprietà Chart per restituire l'oggetto Chart contenuto nell'oggetto ChartObject.

```
Sub ModifyChart1()
     ActiveSheet.ChartObjects("Grafico 1").Activate
     ActiveChart.Type = xlArea
End Sub

Sub ModifyChart2()
     ActiveSheet.ChartObjects("Grafico 1").Chart.Type = xlArea
End Sub
```

Cicli nella collezione ChartObjects

Questo esempio modifica il tipo di ogni grafico incorporato nel foglio attivo. La routine utilizza un ciclo For Each-Next per scorrere tutti gli oggetti presenti nella collezione ChartObjects, per accedere a ciascun oggetto Chart e per modificare la relativa proprietà Type.

```
Sub ChartType()
     Dim cht As ChartObject
     For Each cht In ActiveSheet.ChartObjects
         cht.Chart.Type = xlArea
     Next cht
End Sub
```

La seguente macro esegue la stessa funzione, ma opera su tutti i fogli grafici della cartella di lavoro attiva:

```
Sub ChartType2()
     Dim cht As Chart
     For Each cht In ActiveWorkbook.Charts
         cht.Type = xlArea
     Next cht
End Sub
```

Modificare le proprietà di un grafico

L'esempio seguente cambia il carattere della legenda per tutti i grafici sul foglio attivo. Per farlo, utilizza un ciclo For-Next per elaborare tutti gli oggetti ChartObject:

```
Sub LegendMod()
     Dim chtObj As ChartObject
     For Each chtObj In ActiveSheet.ChartObjects
         With chtObj.Chart.Legend.Font
              .Name = "Calibri"
              .FontStyle = "Bold"
              .Size = 12
         End With
     Next cht
End Sub
```

Notate che l'oggetto Font è contenuto nell'oggetto Legend, il quale è contenuto nell'oggetto Chart, contenuto a sua volta nella collezione ChartObjects. Ora capite perché si parla di *gerarchia di oggetti*?

Applicare formattazioni al grafico

Questo esempio applica diversi tipi di formattazione al grafico attivo. Di nuovo, non è necessario conoscere a memoria tutta la sintassi da usare. Questa particolare routine è stata creata registrando una macro e poi ripulendo il codice per rimuovere tutta la sintassi estranea inserita dal registratore di macro.

```
Sub ChartMods()
     ActiveChart.Type = xlArea
     ActiveChart.ChartArea.Font.Name = "Calibri"
     ActiveChart.ChartArea.Font.FontStyle = "Regular"
     ActiveChart.ChartArea.Font.Size = 9
     ActiveChart.PlotArea.Interior.ColorIndex = xlNone
     ActiveChart.Axes(xlValue).TickLabels.Font.Bold = True
     ActiveChart.Axes(xlCategory).TickLabels.Font.Bold = True
     ActiveChart.Legend.Position = xlBottom
End Sub
```

Prima di eseguire la macro ChartMods è necessario attivare un grafico. Attivate un grafico incorporato facendo clic su di esso. Per attivare un grafico in un foglio grafico, basta attivare il foglio grafico.

Per garantire che un grafico sia selezionato, potete aggiungere un'istruzione per determinare se il grafico è attivo. Ecco la macro modificata, che visualizza un messaggio (e termina) se un grafico non è attivato:

```
Sub ChartMods2()
     If ActiveChart Is Nothing Then
         MsgBox "Attivate un grafico."
         Exit Sub
     End If
     ActiveChart.Type = xlArea
     ActiveChart.ChartArea.Font.Name = "Calibri"
     ActiveChart.ChartArea.Font.FontStyle = "Regular"
     ActiveChart.ChartArea.Font.Size = 9
     ActiveChart.PlotArea.Interior.ColorIndex = xlNone
     ActiveChart.Axes(xlValue).TickLabels.Font.Bold = True
     ActiveChart.Axes(xlCategory).TickLabels.Font.Bold = True
     ActiveChart.Legend.Position = xlBottom
End Sub
```

Ecco ora un'altra versione che utilizza il costrutto With-End With per risparmiare un po' di lavoro alla tastiera e rendere il codice un po' più leggibile. Qualche pagina fa vi ho già parlato della struttura With End-With.

```
Sub ChartMods3()
     If ActiveChart Is Nothing Then
         MsgBox "Attivate un grafico."
         Exit Sub
     End If
     With ActiveChart
         .Type = xlArea
         .ChartArea.Font.Name = "Calibri"
         .ChartArea.Font.FontStyle = "Regular"
         .ChartArea.Font.Size = 9
         .PlotArea.Interior.ColorIndex = xlNone
         .Axes(xlValue).TickLabels.Font.Bold = True
         .Axes(xlCategory).TickLabels.Font.Bold = True
         .Legend.Position = xlBottom
     End With
End Sub
```

Questa breve parte scalfisce a malapena le basi dell'utilizzo di VBA per lavorare sui grafici. Ovviamente c'è molto di più da sapere, ma almeno questa introduzione vi sarà utile per procedere nella giusta direzione.

Suggerimenti per velocizzare VBA

VBA è veloce, ma non sempre lo è abbastanza (i programmi non sono mai abbastanza veloci). In questa parte presento alcuni esempi di programmazione che potete usare per velocizzare le vostre macro.

Disattivare l'aggiornamento dello schermo

Quando si esegue una macro, potete sedervi e controllare sullo schermo tutte le azioni che svolge. Anche se farlo può essere istruttivo, una volta che la macro funziona correttamente, la cosa diventa noiosa e rallenta anche notevolmente le prestazioni della macro. Fortunatamente, potete disabilitare l'aggiornamento dello schermo che normalmente si verifica quando si esegue una macro. Per disattivare l'aggiornamento dello schermo, utilizzate la seguente istruzione:

```
Application.ScreenUpdating = False
```

Se invece desiderate che l'utente veda quello che sta succedendo in qualsiasi fase dell'esecuzione della macro, usate la seguente istruzione per riattivare l'aggiornamento dello schermo:

```
Application.ScreenUpdating = True
```

Per mostrare quale può essere la differenza di velocità, eseguite questa semplice macro, che riempie un intervallo con una serie di numeri:

```
Sub FillRange()
     Dim r as Long, c As Long
     Dim Number as Long
     Number = 0
     For r = 1 To 50
         For c = 1 To 50
              Number = Number + 1
              Cells(r, c).Select
              Cells(r, c).Value = Number
         Next c
     Next r
End Sub
```

Vedete la selezione di ogni cella e l'inserimento del valore, cella per cella. Ora ponete la seguente istruzione all'inizio della routine ed eseguitela di nuovo:

```
Application.ScreenUpdating = False
```

L'intervallo viene riempito molto più rapidamente e l'attività non viene visualizzata, fino a quando la macro non è terminata e l'aggiornamento dello schermo viene (automaticamente) impostato su True.

CONSIGLIO

Quando eseguite il debug del codice, l'esecuzione del programma a volte termina a metà, senza che l'aggiornamento dello schermo venga riattivato. Questo, a volte, fa sì che la finestra dell'applicazione di Excel non risponda come si deve. La via d'uscita da questo stato di congelamento è semplice: tornate al VBE ed eseguite nella finestra Immediata la seguente istruzione:

```
Application.ScreenUpdating = True
```

Disattivare il calcolo automatico

In un foglio di lavoro con molte formule complesse, potreste scoprire che, quando la macro è in esecuzione, potete accelerare notevolmente le cose impostando la modalità di calcolo su manuale. Al termine dell'esecuzione della macro, reimpostate la modalità di calcolo su automatico.

La seguente istruzione imposta la modalità di calcolo Excel su manuale:

```
Application.Calculation = xlCalculationManual
```

Questa istruzione, invece, imposta la modalità di calcolo su automatico:

```
Application.Calculation = xlCalculationAutomatic
```

WARNING

Se il vostro codice utilizza celle con formule, disattivando il calcolo le celle con formule non vengono ricalcolate, a meno che chiediate esplicitamente a Excel di farlo!

Eliminare i fastidiosi messaggi di allarme

Come sapete, una macro può eseguire automaticamente tutta una serie di azioni. In molti casi, potete avviare una macro e prendervi una pausa mentre Excel fa il suo dovere. Alcune operazioni di Excel, tuttavia, visualizzano messaggi che richiedono una risposta umana. Per esempio, se la macro elimina un foglio non vuoto, il codice si arresta bruscamente, mentre Excel attende la risposta al messaggio rappresentato nella Figura 14.4. Questi tipi di messaggi fanno sì che non possiate lasciare Excel incustodito mentre esegue la macro, a meno che non conosciate un certo trucco segreto…

FIGURA 14.4 Potete indicare a Excel di non visualizzare questi tipi di avvisi durante l'esecuzione di una macro.

Il trucco segreto per evitare questi messaggi di avviso consiste nell'inserire nella macro la seguente istruzione VBA:

```
Application.DisplayAlerts = False
```

Excel userà l'opzione predefinita per questi tipi di messaggi. In caso di cancellazione di un foglio, l'operazione predefinita è Elimina (che è proprio quello che desiderate che succeda). Se non siete sicuri di quale sia l'opzione predefinita, fate una prova per vedere quello che accade.

Al termine della routine, Excel reimposterà automaticamente la proprietà DisplayAlerts su True (il suo stato normale). Se dovete riattivare gli avvisi prima che la routine termini, utilizzate questa istruzione:

```
Application.DisplayAlerts = True
```

Semplificare i riferimenti agli oggetti

Come probabilmente già sapete, i riferimenti agli oggetti possono diventare anche molto lunghi. Per esempio, un riferimento completo a un oggetto Range può avere il seguente aspetto:

```
Workbooks("NomeCartella.xlsx").Worksheets("Foglio1") _
     .Range("TassoInteresse")
```

Se la vostra macro utilizza frequentemente questo intervallo, potreste voler creare una variabile oggetto usando il comando Set. Per esempio, la seguente istruzione assegna questo oggetto Range a una variabile oggetto denominata Rate:

```
Set Rate = Workbooks("NomeCartella.xlsx") _
     .Worksheets("Foglio1").Range("TassoInteresse")
```

Dopo aver definito questa variabile oggetto, potete utilizzare la variabile Rate al posto del riferimento completo. Per esempio, potete modificare il valore della cella denominata TassoInteresse così:

```
Rate.Value =.085
```

Questa è molto più facile da digitare (e da comprendere) rispetto alla seguente istruzione:

```
Workbooks("NomeCartella.xlsx").Worksheets("Foglio1"). _
     Range("TassoInteresse") =.085
```

Oltre a semplificare il programma, l'uso delle variabili oggetto accelera notevolmente le macro; a volte la velocità raddoppia!

Dichiarare i tipi di variabile

Di solito non dovete preoccuparvi del tipo di dati che assegnate a una variabile. Excel gestisce tutti i dettagli per voi dietro le quinte. Per esempio, se disponete di una variabile denominata MyVar, potete assegnare un numero di qualsiasi tipo a tale variabile. Potete anche assegnarle una stringa di testo, più avanti nella routine.

RICORDA!

Se desiderate che le vostre routine siano eseguite il più velocemente possibile (e se volete evitare alcuni problemi potenzialmente sgradevoli), comunicate a Excel quale tipo di dati intendete assegnare a ciascuna delle variabili. Questo concetto è noto come *dichiarazione* del tipo di una variabile (per i dettagli fate riferimento al Capitolo 7). Adottate l'abitudine di dichiarare tutte le variabili che usate.

In generale, è necessario utilizzare il tipo di dati che richiede il minor numero di byte, ma che comunque permetta di gestire tutti i dati ad esso assegnati. Quando VBA opera sui dati, la velocità di esecuzione dipende dal numero grezzo di byte che VBA deve manipolare. In altre parole, meno byte vengono utilizzati, più velocemente VBA potrà leggere e manipolare i dati. Un'eccezione è rappresentata dal tipo di dati Integer. Se la velocità è critica, utilizzate piuttosto il tipo di dati Long.

Se utilizzate una variabile oggetto (come descritto nel paragrafo precedente), potete dichiararla come un tipo specifico di oggetto. Ecco un esempio:

```
Dim Rate as Range
Set Rate = Workbooks("NomeCartella.xlsx") _
      .Worksheets("Foglio1").Range("TassoInteresse")
```

Usare la struttura With-End With

Avete bisogno di impostare un certo numero di proprietà di un oggetto? Il codice funziona più velocemente se usate una struttura With-End With. Un ulteriore vantaggio è che migliorerete la leggibilità del vostro codice.

Il seguente codice non usa With-End With:

```
Selection.HorizontalAlignment = xlCenter
Selection.VerticalAlignment = xlCenter
Selection.WrapText = True
Selection.Orientation = 0
Selection.ShrinkToFit = False
Selection.MergeCells = False
```

Ecco lo stesso codice, riscritto per utilizzare With-End With:

```
With Selection
      .HorizontalAlignment = xlCenter
      .VerticalAlignment = xlCenter
      .WrapText = True
      .Orientation = 0
      .ShrinkToFit = False
      .MergeCells = False
End With
```

Come potete vedere, questo codice dice a Excel che le prossime istruzioni si applicano all'oggetto Selection. Questo è il motivo per cui tutte le istruzioni all'interno della struttura With-End With iniziano con un punto. State semplicemente evitando di ripetere sempre lo stesso oggetto (in questo caso, l'oggetto Selection). Ciò non solo significa meno fatica alla tastiera, ma anche meno elaborazione per Excel.

4

Comunicare con gli utenti

IN QUESTA PARTE...

- Risparmiare tempo in vari modi utilizzando metodi alternativi di comunicazione.
- Imparare a sviluppare finestre di dialogo personalizzate (note anche come UserForm).
- Come utilizzare i controlli della finestra di dialogo, per esempio pulsanti, caselle di riepilogo e caselle di controllo.
- Tanti trucchi e suggerimenti utili per creare finestre di dialogo personalizzate.
- Adattare l'interfaccia utente di Excel per facilitare l'esecuzione delle macro.

Capitolo 15

Semplici finestre di dialogo

IN QUESTO CAPITOLO

- **Risparmiare tempo con una delle numerose alternative alle UserForm**
- **Le funzioni InputBox e MsgBox per ottenere informazioni dall'utente**
- **Chiedere all'utente un nome e un percorso di file**
- **Chiedere all'utente il nome di una cartella**
- **Scrivere codice VBA per eseguire comandi della barra multifunzione che aprono finestre di dialogo incorporate di Excel**

Non potete utilizzare Excel molto a lungo senza avere a che fare con le finestre di dialogo. Sembrano apparire continuamente. Excel, come la maggior parte dei programmi Windows, utilizza le finestre di dialogo per ottenere informazioni, chiarire comandi e visualizzare messaggi. Se sviluppate macro VBA, potete creare finestre di dialogo personalizzate che funzionano esattamente come quelle standard di Excel. In VBA le finestre di dialogo personalizzate sono chiamate UserForm.

Questo capitolo non dice nulla sulla creazione di UserForm. Descrive piuttosto alcune tecniche utili che potreste sfruttare *senza ricorrere* alle UserForm. I Capitoli da 16 a 18, al contrario, sono dedicati alle UserForm.

Alternative alle UserForm

Alcune macro VBA si comportano allo stesso modo ogni volta che le eseguite. Per esempio, potete sviluppare una macro che inserisce un elenco di dipendenti in un intervallo di fogli di lavoro. Questa macro produce sempre lo stesso risultato e non richiede alcun input da parte dell'utente.

Potreste sviluppare altre macro, tuttavia, che in varie circostanze si comportano diversamente o che offrono delle opzioni dell'utente. In questi casi, la macro potrebbe trarre vantaggio da una finestra di dialogo personalizzata. Una finestra di dialogo personalizzata fornisce un mezzo semplice per ottenere informazioni dall'utente. La macro, poi, utilizzerà tali informazioni per determinare quello che deve fare.

Le UserForm possono essere abbastanza utili, ma crearle richiede tempo. Prima di approfondire l'argomento della creazione di UserForm, nel prossimo capitolo, è bene conoscere alcune alternative potenzialmente utili.

Il linguaggio VBA consente di visualizzare diversi tipi di finestre di dialogo che potete utilizzare al posto di una UserForm scritta a mano. Potete personalizzare queste finestre di dialogo incorporate in alcuni modi, ma certamente non offrono le opzioni disponibili in una UserForm. In alcuni casi, tuttavia, sono esattamente ciò di cui c'è bisogno.

In questo capitolo parleremo dei seguenti argomenti.

- La funzione MsgBox.
- La funzione InputBox.
- Il metodo GetOpenFilename.
- Il metodo GetSaveAsFilename.
- Il metodo FileDialog.

Inoltre, in questo capitolo, impareremo a utilizzare VBA per visualizzare alcune finestre di dialogo incorporate di Excel, ovvero le finestre di dialogo utilizzate da Excel per chiedere informazioni all'utente.

La funzione MsgBox

A questo punto, ormai probabilmente avrete familiarizzato con la funzione MsgBox, presente in molti degli esempi dei capitoli precedenti. La funzione MsgBox, i cui argomenti principali sono elencati nella Tabella 15.1, è utile per

visualizzare informazioni e per ottenere semplici input da parte dell'utente. È in grado di accettare l'input dell'utente perché è una funzione. Una funzione, come ho già detto, restituisce un valore. Nel caso della funzione MsgBox, utilizza una finestra di dialogo per ottenere il valore restituito. Continuate a leggere per scoprire come funziona, esattamente.

TABELLA 15.1 Argomenti della funzione MsgBox

Argomento	Che cosa influenza
Prompt	Il testo visualizzato da Excel nella finestra.
Pulsanti	Un numero che specifica quali pulsanti (insieme a quale icona) compaiono nella finestra (opzionale).
Titolo	Il testo che appare nella barra del titolo della finestra (opzionale).

Ecco una versione semplificata della sintassi della funzione MsgBox:

```
MsgBox(messaggio[, pulsanti][, titolo])
```

Visualizzare una semplice finestra

Potete utilizzare la funzione MsgBox in due modi.

- » Per mostrare un messaggio all'utente: in questo caso, non vi interessa il risultato restituito dalla funzione.
- » Per ottenere una risposta dall'utente: in questo caso, il risultato restituito dalla funzione è importante. Il risultato dipende dal pulsante su cui l'utente fa clic.

Se utilizzate la funzione MsgBox da sola, potete tralasciare le parentesi attorno agli argomenti. L'esempio seguente mostra un messaggio e non restituisce alcun risultato. Dopo la visualizzazione del messaggio, il codice si arresta finché l'utente non fa clic su OK.

```
Sub MsgBoxDemo()
     MsgBox "Fate clic su OK per iniziare a stampare."
'   Sheets("Risultati"). PrintOut
End Sub
```

La Figura 15.1 mostra l'aspetto di questa finestra MsgBox. In questo caso, la stampa inizia quando l'utente fa clic su OK. Vi siete accorti che non c'è modo di annullare la stampa? Il prossimo paragrafo descrive come risolvere il problema.

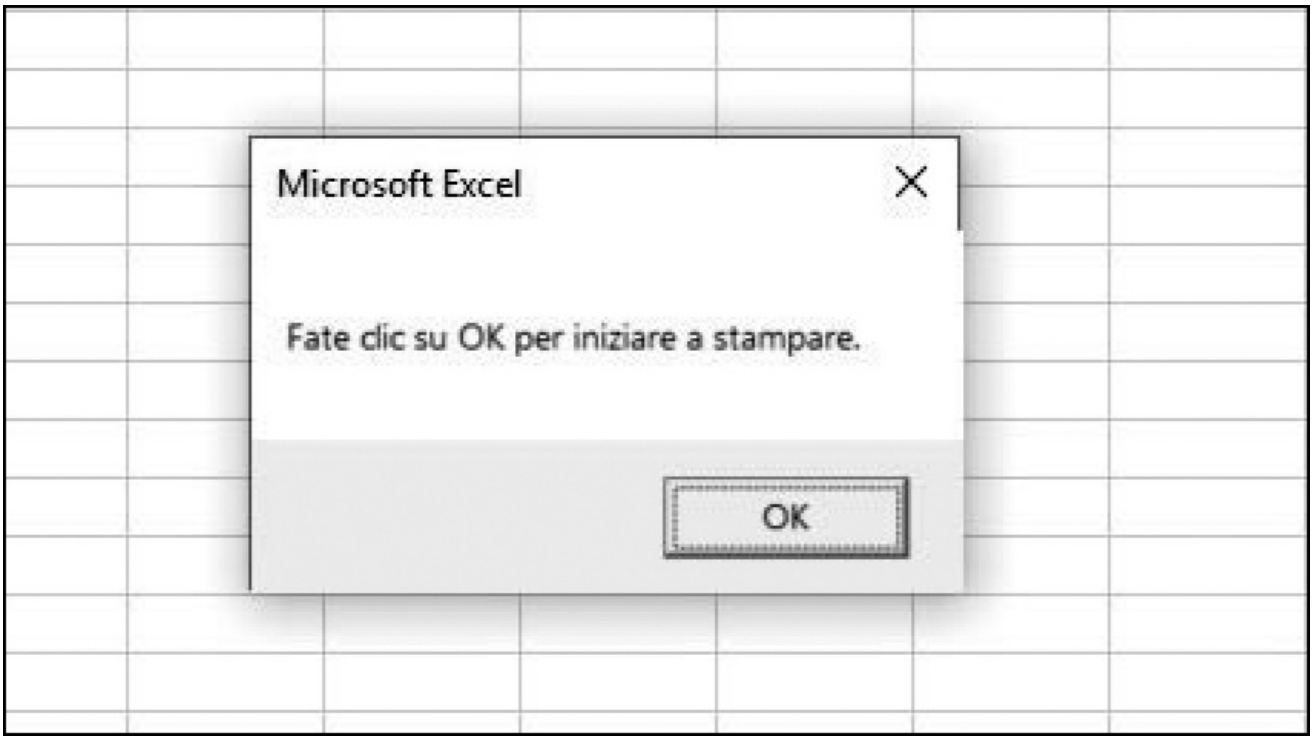

FIGURA 15.1 Una semplice finestra MsgBox.

Ottenere una risposta da una finestra MsgBox

Se visualizzate una finestra MsgBox che ha qualcosa di più di un semplice pulsante OK, probabilmente vorrete sapere su quale pulsante ha fatto clic l'utente. Siete fortunati. La funzione MsgBox restituisce un valore che rappresenta il

pulsante selezionato. Quindi potete assegnare il risultato della funzione MsgBox a una variabile.

Il codice seguente utilizza alcune costanti predefinite (descritte nella Tabella 15.2) che facilitano il lavoro con i valori restituiti da MsgBox:

```
Sub GetAnswer()
     Dim Ans As Long
     Ans = MsgBox("Avviare la stampa?", vbYesNo)
     Select Case Ans
         Case vbYes
              ActiveSheet.PrintOut
         Case vbNo
              MsgBox "Stampa annullata"
     End Select
End Sub
```

La Figura 15.2 mostra il suo aspetto. Quando si esegue questa routine, alla variabile Ans viene assegnato il valore vbYes oppure vbNo, a seconda del pulsante su cui l'utente ha fatto clic. Poi l'istruzione Select Case utilizza il valore Ans per determinare quale azione deve eseguire il codice.

TABELLA 15.2 Costanti utilizzate dalla funzione MsgBox

Costante	Valore	Che cosa fa
vbOKOnly	0	Visualizza il solo pulsante OK.
vbOKCancel	1	Visualizza i pulsanti OK e Annulla.
vbAbortRetryIgnore	2	Visualizza i pulsanti Annulla, Riprova e Ignora.
vbYesNoCancel	3	Visualizza i pulsanti Sì, No e Annulla.
vbYesNo	4	Visualizza i pulsanti Sì e No.
vbRetryCancel	5	Visualizza i pulsanti Riprova e Annulla.
vbCritical	16	Visualizza l'icona del messaggio critico.
vbQuestion	32	Visualizza l'icona del punto interrogativo.
vbExclamation	48	Visualizza l'icona del punto esclamativo.
vbInformation	64	Visualizza l'icona del messaggio informativo.
vbDefaultButton1	0	Il pulsante predefinito è il primo.
vbDefaultButton2	256	Il pulsante predefinito è il secondo.
vbDefaultButton3	512	Il pulsante predefinito è il terzo.
vbDefaultButton4	768	Il pulsante predefinito è il quarto.

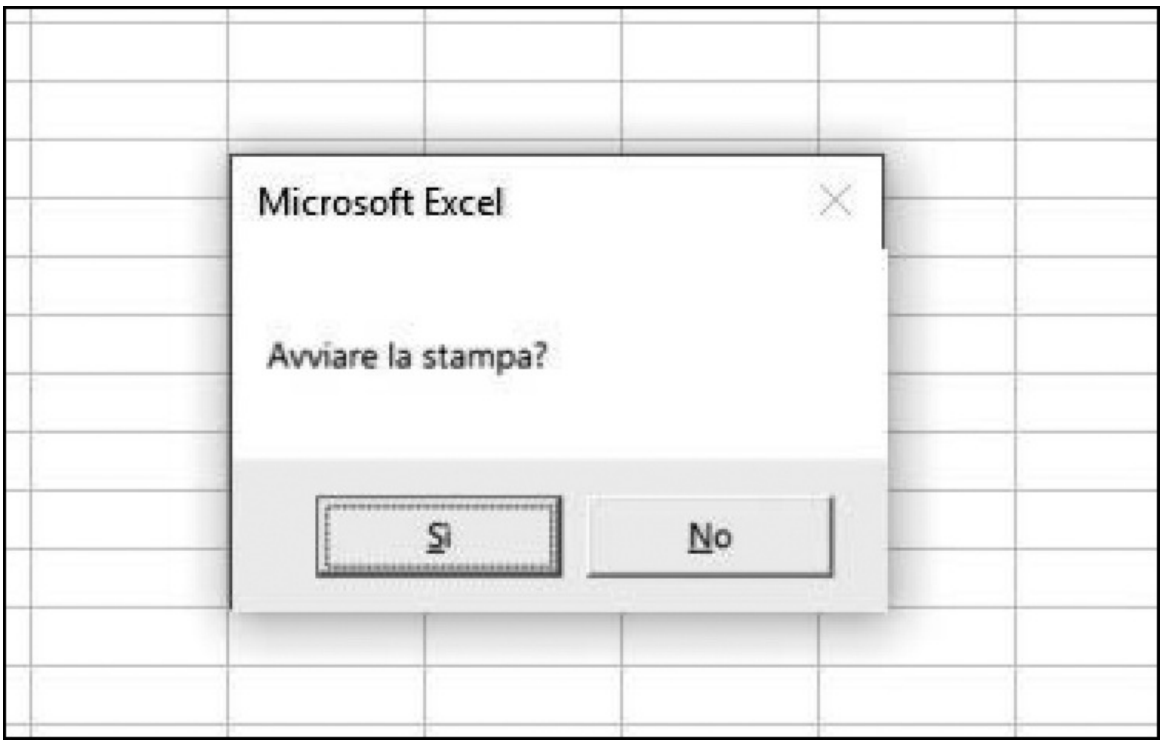

FIGURA 15.2 Una semplice finestra MsgBox, con due pulsanti.

Potete anche utilizzare direttamente il risultato della funzione MsgBox senza utilizzare una variabile, come mostra il

seguente esempio:

```
Sub GetAnswer2()
     If MsgBox("Avviare la stampa?", vbYesNo) = vbYes Then
          '   ...[codice da eseguire dopo un clic su Sì]...
     Else
          '   ...[codice da eseguire in ogni altro caso]...
     End If
End Sub
```

Personalizzare le finestre MsgBox

La flessibilità dell'argomento per i pulsanti facilita la personalizzazione delle finestre MsgBox. Potete scegliere quali pulsanti visualizzare, determinare se visualizzare un'icona e decidere quale pulsante rappresenta l'impostazione predefinita (il pulsante scelto se l'utente preme semplicemente Invio).

Per utilizzare come argomento una o più di queste costanti, è sufficiente collegarle con un operatore +. Per esempio, per visualizzare una finestra MsgBox con i pulsanti Sì e No e un'icona a punto esclamativo, utilizzate, come secondo argomento di MsgBox, la seguente espressione:

```
vbYesNo + vbExclamation
```

Oppure, se davvero preferite rendere il vostro codice incomprensibile, usate il valore 52 (ovvero 4 + 48).

L'esempio seguente utilizza una combinazione di costanti per visualizzare una finestra MsgBox con un pulsante Sì, un pulsante No (vbYesNo) e un'icona a punto interrogativo (vbQuestion). La costante vbDefaultButton2 definisce il secondo pulsante (No) come pulsante predefinito, ovvero il pulsante selezionato se l'utente preme semplicemente Invio. Per semplicità, potete assegnare queste costanti alla variabile Config e poi utilizzare Config come secondo argomento nella funzione MsgBox:

```
Sub GetAnswer3()
     Dim Config As Long
     Dim Ans As Integer
     Config = vbYesNo + vbQuestion + vbDefaultButton2
     Ans = MsgBox("Generare il report mensile?", Config)
     If Ans = vbYes Then RunReport
End Sub
```

La Figura 15.3 mostra la finestra MsgBox visualizzata in Excel quando si esegue la routine GetAnswer3. Se l'utente fa clic sul pulsante Sì, la routine esegue la routine denominata RunReport (non mostrata). Se l'utente fa clic sul pulsante No (o preme Invio), la routine termina senza alcuna azione. Poiché la routine omette l'argomento per il titolo nella funzione MsgBox, Excel utilizza il titolo predefinito: Microsoft Excel.

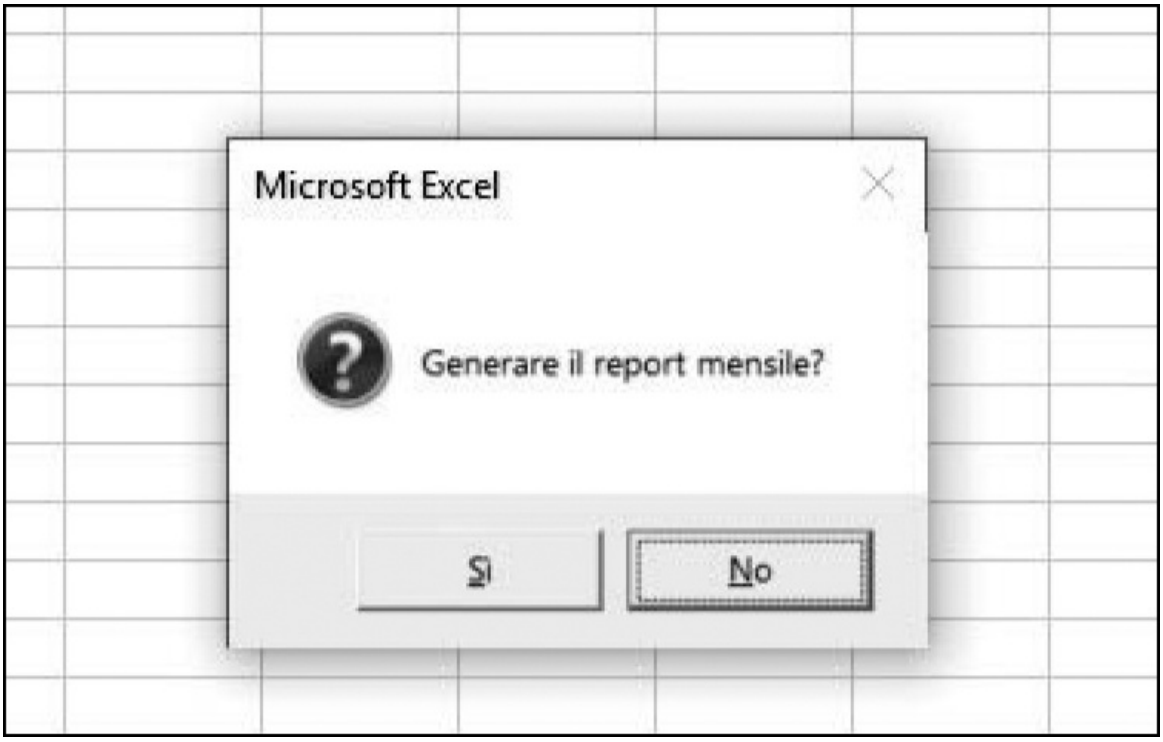

FIGURA 15.3 L'argomento dei pulsanti della funzione MsgBox determina ciò che viene visualizzato nella finestra del messaggio.

La seguente routine rappresenta un altro esempio di utilizzo della funzione MsgBox:

```
Sub GetAnswer4()
     Dim Msg As String, Title As String
     Dim Config As Integer, Ans As Integer
     Msg = "Volete generare il report mensile?"
     Msg = Msg & vbNewLine & vbNewLine
     Msg = Msg & "L'elaborazione del report mensile "
     Msg = Msg & "richiederà circa 15 minuti. Verrà"
     Msg = Msg & "generato un report di 30 pagine per "
     Msg = Msg & "tutte le sedi commerciali per "
     Msg = Msg & "il mese corrente."
     Title = "XYZ Marketing"
     Config = vbYesNo + vbQuestion
     Ans = MsgBox(Msg, Config, Title)
     If Ans = vbYes Then RunReport
End Sub
```

Questo esempio presenta un modo efficace per specificare, in una finestra MsgBox, un messaggio più lungo. Qui, la variabile (Msg) e l'operatore di concatenamento (&) costruiscono il messaggio con una lunga serie di istruzioni. La costante vbNewLine inserisce un carattere di fine riga che avvia una nuova riga (utilizzatela due volte per inserire una riga vuota). L'argomento Title visualizza un apposito titolo nella finestra.

La Figura 15.4 mostra la finestra MsgBox visualizzata in Excel quando si esegue questa routine.

Gli esempi precedenti utilizzano delle costanti (come vbYes e vbNo) per controllare il valore restituito dalla funzione MsgBox. La Tabella 15.3 elenca alcune altre costanti.

FIGURA 15.4 Questa finestra di dialogo, visualizzata dalla funzione MsgBox, mostra un titolo, un'icona e due pulsanti.

TABELLA 15.3 **Costanti utilizzate per il valore restituito dalla funzione MsgBox**

Costante	Valore	Che cosa significa
vbOK	1	L'utente ha fatto clic su OK.
vbCancel	2	L'utente ha fatto clic su Annulla.
vbAbort	3	L'utente ha fatto clic su Annulla.
vbRetry	4	L'utente ha fatto clic su Riprova.
vbIgnore	5	L'utente ha fatto clic su Ignora.
vbYes	6	L'utente ha fatto clic su Sì.
vbNo	7	L'utente ha fatto clic su No.

E questo è tutto ciò che occorre sapere sulla funzione MsgBox. Usate questa funzione con parsimonia. Di solito non c'è motivo di visualizzare finestre di messaggio che non servono a nulla. Per esempio, gli utenti tendono ad arrabbiarsi se ogni santo giorno gli propinate una finestra con un messaggio del tipo: "Buongiorno. Grazie per aver caricato la cartella di lavoro Budget Projection".

La funzione InputBox

La funzione InputBox è utile per ottenere una singola informazione digitata dall'utente. Tale informazione potrebbe essere un valore, una stringa di testo o perfino un intervallo di celle. È una buona alternativa allo sviluppo di una UserForm quando è necessario ottenere un solo valore.

La sintassi di InputBox

Ecco una versione semplificata della sintassi della funzione InputBox:

```
InputBox(messaggio[, titolo][, default])
```

La Tabella 15.4 mostra gli argomenti chiave per la funzione InputBox.

TABELLA 15.4 Argomenti della funzione InputBox

Argomento	Che cosa influenza
Messaggio	Il testo visualizzato nella InputBox.
Titolo	Il testo visualizzato nella barra del titolo della InputBox (opzionale).
Default	Il valore predefinito per la InputBox (opzionale).

Un esempio di InputBox

Ecco un'istruzione che mostra come utilizzare la funzione InputBox:

```
TheName = InputBox("Qual è il suo nome?", "Saluti")
```

Quando si esegue questa istruzione VBA, Excel visualizza la finestra di dialogo rappresentata nella Figura 15.5. Notate che questo esempio utilizza solo i primi due argomenti e non fornisce un valore predefinito. Quando l'utente inserisce un valore e fa clic su OK, il codice assegna il valore letto alla variabile TheName.

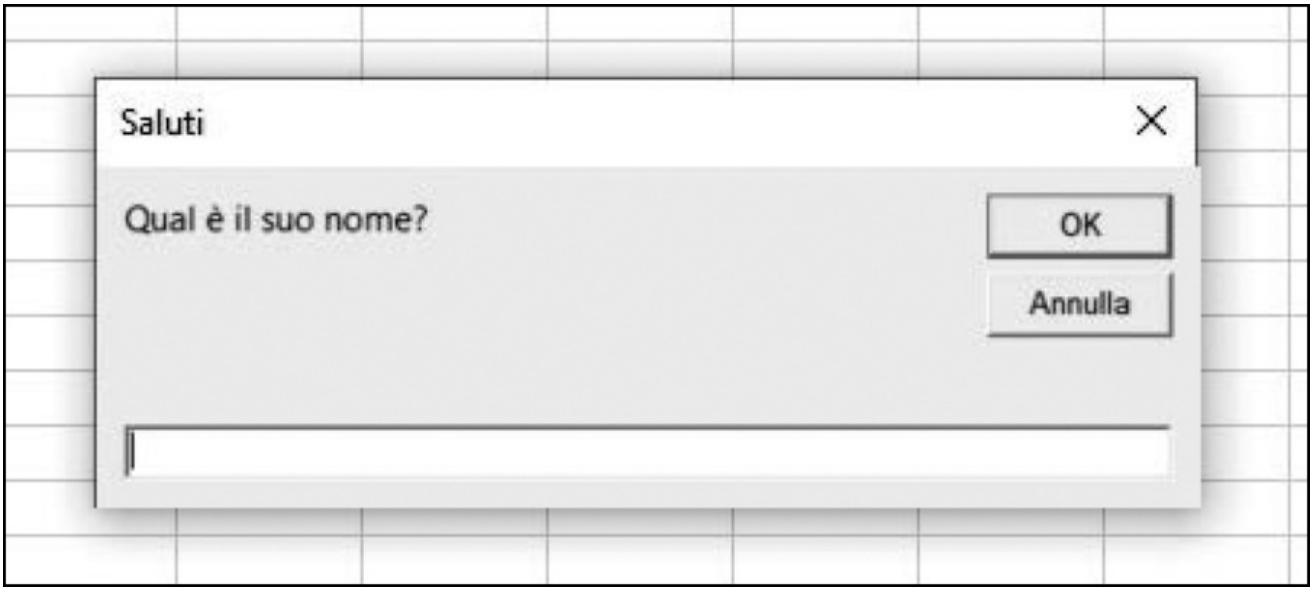

FIGURA 15.5 La funzione InputBox visualizza questa finestra di dialogo.

L'esempio seguente utilizza il terzo argomento e fornisce un valore predefinito. Il valore predefinito è il nome dell'utente, memorizzato da Excel (la proprietà UserName dell'oggetto Application).

```
Sub GetName()
     Dim TheName As String
     TheName = InputBox("Qual è il suo nome?", _
         "Saluti", Application.UserName)
End Sub
```

InputBox visualizza sempre un pulsante Annulla. Se l'utente fa clic su Annulla, la funzione InputBox restituisce una stringa vuota.

RICORDA!

La funzione InputBox restituisce sempre una stringa, quindi se è necessario richiedere un valore, il codice deve eseguire un controllo aggiuntivo. Nell'esempio seguente viene utilizzata la funzione InputBox per ottenere un numero. Si usa la funzione IsNumeric per verificare se la stringa fornita è effettivamente un numero. Se la stringa contiene un numero, tutto va bene. Se ciò che ha specificato l'utente non può essere interpretato come un numero, il codice apre una finestra MsgBox.

```
Sub AddSheets()
     Dim Prompt As String
     Dim Caption As String
     Dim DefValue As Long
     Dim NumSheets As String

     Prompt = "Quanti fogli occorre aggiungere?"
     Caption = "Domanda…"
     DefValue = 1
     NumSheets = InputBox(Prompt, Caption, DefValue)

     If NumSheets = "" Then Exit Sub ';Canceled
     If IsNumeric(NumSheets) Then
         If NumSheets > 0 Then Sheets.Add Count:=NumSheets
     Else
         MsgBox "Numero non valido"
     End If
End Sub
```

La Figura 15.6 mostra la finestra di dialogo prodotta da questa routine.

Un altro tipo di InputBox

Le informazioni presentate finora in questo paragrafo si applicano alla *funzione* InputBox. Microsoft a volte sembra amare la confusione, quindi vi mette a disposizione anche un *metodo* InputBox, dell'oggetto Application.

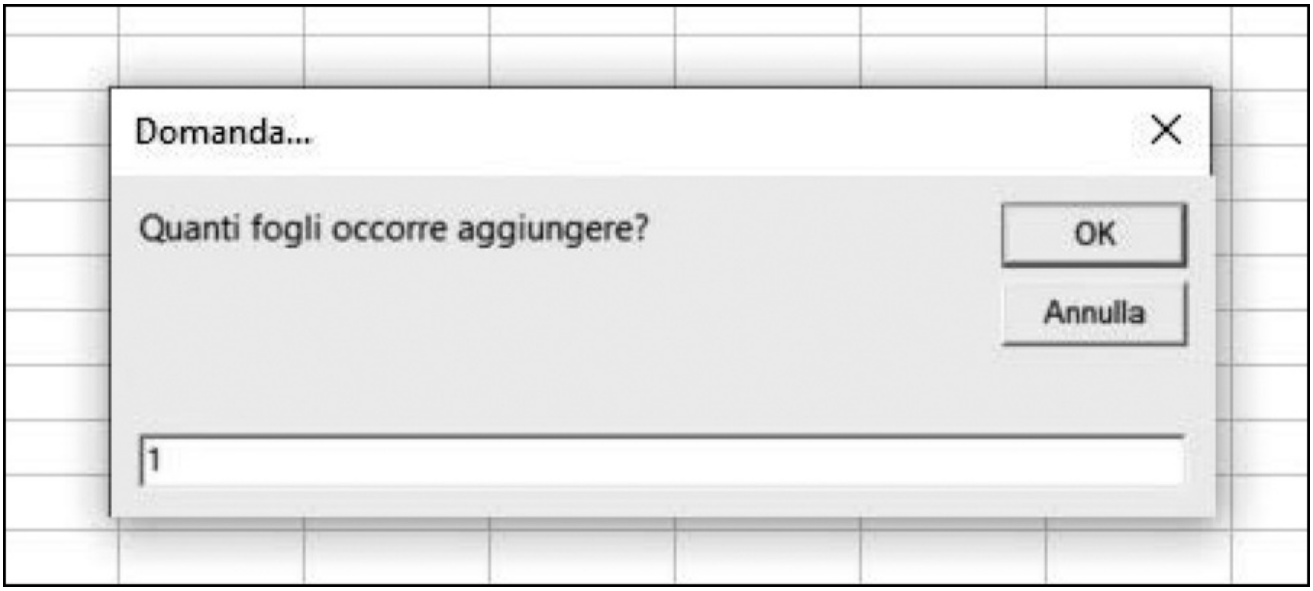

FIGURA 15.6 Un altro esempio di utilizzo della funzione InputBox.

Un grande vantaggio del metodo InputBox di Application è il fatto che il codice può chiedere anche di selezionare un intervallo. L'utente può quindi selezionare l'intervallo nel foglio di lavoro evidenziando le celle. Ecco un breve esempio che chiede all'utente di selezionare un intervallo:

```
Sub GetRange()
     Dim Rng As Range
     On Error Resume Next
     Set Rng = Application.InputBox _
         (prompt:="Selezionate un intervallo:", Type:=8)
     If Rng Is Nothing Then Exit Sub
     MsgBox "Avete selezionato l'intervallo " & Rng.Address
End Sub
```

La Figura 15.7 mostra il risultato.

FIGURA 15.7 Uso del metodo InputBox di Application per ottenere un intervallo.

In questo semplice esempio, il codice non fa altro che indicare all'utente l'indirizzo dell'intervallo selezionato. Nella vita reale, il vostro codice farebbe qualcosa di più utile con l'intervallo selezionato. Una cosa carina di questo esempio è che Excel si occupa anche della gestione degli errori. Se inserite qualcosa che non sia un intervallo, Excel ve lo comunica e vi consente di riprovare.

Il metodo Application.InputBox è simile alla funzione InputBox, ma presenta anche alcune differenze. Nella Guida trovate tutti i dettagli.

Il metodo GetOpenFilename

Se la vostra routine VBA deve richiedere all'utente un nome di file, *potete* certamente utilizzare la funzione InputBox e fare in modo che l'utente lo digiti a mano. Ma, in genere, una casella di input non è lo strumento migliore per questo compito, in quanto la maggior parte degli utenti fatica a ricordare percorsi, simboli, nomi ed estensioni di file. In altre parole, è fin troppo facile sbagliarsi quando si digita un nome di file.

Per una migliore soluzione a questo problema, utilizzate il metodo GetOpenFilename dell'oggetto Application, che garantisce che il codice usi un nome di file valido, completo di percorso. Il metodo GetOpenFilename visualizza la familiare finestra di dialogo Apri (la stessa identica finestra che Excel visualizza quando si sceglie File ➪ Apri ➪ Sfoglia).

RICORDA!

Nonostante il nome, il metodo GetOpenFilename non *apre* il file specificato. Questo metodo restituisce semplicemente il nome del file selezionato dall'utente, sotto forma di stringa. Quindi potete scrivere tutto il codice che volete per fare quello che desiderate con il nome del file.

La sintassi del metodo GetOpenFilename

La sintassi del metodo GetOpenFilename è la seguente:

```
object.GetOpenFilename ([FileFilter], [FilterIndex],
     [Title],[ButtonText], [MultiSelect])
```

Il metodo GetOpenFilename accetta gli argomenti facoltativi mostrati nella Tabella 15.5.

TABELLA 15.5 Argomenti del metodo GetOpenFilename

Argomento	Che cosa fa

FileFilter	Determina i tipi di file visualizzati nella finestra di dialogo (per esempio, *.txt). Potete specificare diversi filtri tra cui l'utente può scegliere.
FilterIndex	Determina quali filtri di file la finestra di dialogo visualizza come impostazione predefinita.
Title	Specifica la didascalia per la barra del titolo della finestra di dialogo.
ButtonText	Ignorato (utilizzato solo dalla versione Macintosh di Excel).
MultiSelect	Se è True, l'utente può selezionare più file.

Un esempio d'uso di GetOpenFilename

L'argomento FileFilter determina che cosa viene visualizzato nell'elenco a discesa Tipo di file della finestra di dialogo. Questo argomento è costituito da coppie formate da una stringa per i filtri di file seguita dalla specifica dei caratteri jolly, con una virgola a separare ogni parte e coppia. Se viene omesso, questo argomento assume il seguente valore:

```
Tutti i file (*.*), *.*
```

Notate che questa stringa è composta da due parti, separate da una virgola:

```
Tutti i file (*.*)
```

e

```
*.*
```

La prima parte di questa stringa è il testo visualizzato nell'elenco a discesa Tipi di file. La seconda parte determina i file visualizzati nella finestra di dialogo. Per esempio, *. * mostra tutti i file.

Il codice nell'esempio seguente apre una finestra di dialogo che richiede all'utente un nome di file. La routine definisce cinque filtri di file. Notate che la sequenza di continuazione della riga imposta la variabile Filter; questo aiuta a semplificare un argomento altrimenti complicato.

```
Sub GetImportFileName()
     Dim Finfo As String
     Dim FilterIndex As Long
     Dim Title As String
     Dim FileName As Variant

'   Imposta i filtri di file
       FInfo = "File di testo (*.txt),*.txt," & _
                 "File Lotus (*.prn),*.prn," & _
                 "File CSV (*.csv),*.csv," & _
                 "File ASCII (*.asc),*.asc," & _
                 "Tutti i file (*.*),*.*"
'   Visualizza per default *.*
     FilterIndex = 5

'   Imposta il titolo della finestra di dialogo
     Title = "Selezionate il file da importare"

'   Legge il nome del file
     FileName = Application.GetOpenFilename (FInfo, _
       FilterIndex, Title)

'   Mostra le informazioni richieste
     If FileName = False Then
         MsgBox "Non è stato selezionato alcun file."
     Else
         MsgBox "Avete selezionato " & FileName
     End If
End Sub
```

La Figura 15.8 mostra la finestra di dialogo di Excel visualizzata eseguendo questa routine. L'aspetto può variare a seconda della versione di Windows utilizzata e delle opzioni di visualizzazione impostate.

In un'applicazione reale, dovreste fare qualcosa di più significativo con il nome del file. Per esempio, potreste volerlo aprire usando un'istruzione come la seguente:

```
Workbooks.Open FileName
```

PER I PIÙ CURIOSI

Notate che la variabile FileName è di tipo Variant. Se l'utente fa clic su Annulla, quella variabile contiene un valore booleano (False). Altrimenti, FileName è una stringa. Pertanto, l'utilizzo di un tipo di dati Variant gestisce entrambe le possibilità.

FIGURA 15.8 Il metodo GetOpenFilename apre una finestra di dialogo personalizzabile e restituisce il percorso e il nome del file selezionato. Non apre il file.

Il metodo GetSaveAsFilename

Il metodo GetSaveAsFilename di Excel funziona esattamente come il metodo GetOpenFilename, ma visualizza la finestra di dialogo Salva con nome di Excel. Il metodo GetSaveAsFilename ottiene un percorso e un nome di file dall'utente, ma non fa nulla con quelle informazioni. Spetta a voi scrivere il codice che salva effettivamente il file. La sintassi di questo metodo è la seguente:

```
object.GetSaveAsFilename ([InitialFilename], [FileFilter], [FilterIndex],[Title], [ButtonText])
```

Il metodo GetSaveAsFilename accetta gli argomenti elencati nella Tabella 15.6, che sono tutti facoltativi.

TABELLA 15.6 **Argomenti del metodo GetSaveAsFilename**

Argomento	Che cosa fa
InitialFileName	Specifica un nome di file predefinito visualizzato nella casella Nome file.
FileFilter	Determina i tipi di file visualizzati in Excel nella finestra di dialogo (per esempio, *.txt). Potete specificare diversi filtri, fra i quali l'utente può scegliere.
FilterIndex	Determina quale dei filtri di file viene visualizzato in Excel come impostazione predefinita.
Title	Definisce una didascalia per la barra del titolo della finestra di dialogo.

Ottenere un nome di cartella

A volte, non è necessario ottenere il nome di un file; avete solo bisogno di ottenere il nome di una cartella. Se questo è il caso, l'oggetto FileDialog è proprio perfetto.

La seguente routine mostra una finestra di dialogo che consente all'utente di selezionare una directory. Il nome della directory selezionata (o "Annullata") viene poi visualizzato dalla funzione MsgBox.

```
Sub GetAFolder()
     With Application.FileDialog(msoFileDialogFolderPicker)
         .InitialFileName = Application.DefaultFilePath & "\"
         .Title = "Selezionare una posizione per il backup"
         .Show
         If. SelectedItems.Count = 0 Then
```

```
            MsgBox "Annullata"
        Else
            MsgBox.SelectedItems(1)
        End If
    End With
End Sub
```

L'oggetto FileDialog consente di specificare la directory iniziale indicando un valore per la proprietà InitialFileName. In questo caso, il codice utilizza come directory iniziale il percorso predefinito per i file di Excel.

Visualizzare le finestre di dialogo di Excel

Il linguaggio VBA permette anche di simulare i comandi di Excel. Considerate, per esempio, la seguente istruzione VBA:

```
Range("A1:A12").Name = "NomiMesi"
```

L'esecuzione di questa istruzione VBA equivale a selezionare il comando Formule ➪ Nomi definiti ➪ Definisci nome per visualizzare la finestra di dialogo Nuovo nome, digitare NomiMesi nella casella Nome e poi A1: A12 nella casella Riferito a e fare clic su OK.

Quando si esegue l'istruzione VBA, la finestra di dialogo Nuovo nome non viene visualizzata. Questo è quasi sempre quello che desiderate che accada; non volete che le finestre di dialogo compaiano fugacemente sullo schermo mentre viene eseguita la macro.

In alcuni casi, tuttavia, potreste volere che il codice visualizzi una delle numerose finestre di dialogo di Excel e che consenta all'utente di fare le sue scelte in tale finestra di dialogo. Potete farlo utilizzando VBA per eseguire un comando della barra multifunzione. Ecco un esempio che mostra la finestra di dialogo Nuovo nome. L'indirizzo nella casella Riferito a rappresenta l'intervallo selezionato dopo aver eseguito il comando (vedi la Figura 15.9).

```
Application.CommandBars.ExecuteMso "NameDefine"
```

RICORDA!

Il codice VBA non può ottenere alcuna informazione dalla finestra di dialogo. Per esempio, se si esegue il codice per visualizzare la finestra di dialogo Nuovo nome, il codice non può ottenere il nome inserito dall'utente o l'intervallo che viene chiamato.

ExecuteMso è un metodo dell'oggetto CommandBars e accetta un solo argomento: un parametro idMso che rappresenta un controllo della barra multifunzione. Sfortunatamente, questi parametri non sono elencati nella Guida. E poiché la barra multifunzione non esiste da sempre, il codice che utilizza il metodo ExecuteMso non è compatibile con le versioni di Excel precedenti la 2007.

FIGURA 15.9 Visualizzazione di una finestra di dialogo di Excel tramite VBA.

Ecco un altro esempio di utilizzo del metodo ExecuteMso. Questa istruzione visualizza la scheda Carattere della finestra di dialogo Formato celle:

```
Application.CommandBars.ExecuteMso "FormatCellsFontDialog"
```

Se tentate di visualizzare una finestra di dialogo di Excel in un contesto errato, otterrete un messaggio d'errore. Per esempio, ecco un'istruzione che mostra la scheda Numero della finestra di dialogo Formato celle:

```
Application.CommandBars.ExecuteMso "NumberFormatsDialog"
```

Se si esegue questa istruzione quando non è appropriata (per esempio, se avete selezionato una forma), Excel visualizza un messaggio d'errore, perché quella finestra di dialogo è appropriata solo per le celle del foglio di lavoro.

Excel offre migliaia di comandi. Come trovare quello che vi serve? Un modo consiste nell'utilizzare la scheda Personalizzazione barra multifunzione della finestra di dialogo Opzioni di Excel. Il modo rapido per arrivarci consiste nel fare clic destro su un controllo della barra multifunzione e scegliere dal menu contestuale il comando Personalizza barra multifunzione. Il riquadro a sinistra elenca praticamente ogni comando disponibile in Excel. Trovate il comando che vi serve e passateci sopra con il puntatore del mouse: vi verrà mostrato il suo nome (vi interessa la parte tra parentesi). La Figura 15.10 ne mostra un esempio.

FIGURA 15.10 Utilizzo della scheda Personalizzazione barra multifunzione per identificare il nome di un comando.

Capitolo 16

Concetti di base sulle UserForm

IN QUESTO CAPITOLO

- **Quando usare le UserForm**
- **Che cosa sono gli oggetti UserForm**
- **Visualizzare una UserForm**
- **Creare una UserForm che funziona con una macro**

Una UserForm è utile se la macro VBA deve fermarsi per ottenere informazioni dall'utente. Per esempio, la vostra macro potrebbe avere alcune opzioni che possono dover essere specificate in una UserForm. Se sono richieste solo poche informazioni (per esempio, una risposta Sì/No o una stringa di testo), tale lavoro può essere svolto impiegando una delle tecniche illustrate nel Capitolo 15. Ma se è necessario ottenere informazioni più articolate, occorre creare una UserForm. Questo è l'argomento di questo capitolo.

Quando utilizzare una UserForm

Questo paragrafo descrive le situazioni in cui è preferibile impiegare una UserForm. La seguente macro modifica il testo di ogni cella selezionata, ponendolo in lettere maiuscole. Lo fa utilizzando la funzione UCase di VBA.

```
Sub ChangeCase()
     Dim WorkRange As Range, cell As Range

'   Esce se non è selezionato un intervallo
     If TypeName(Selection) &lt;&gt; "Range" Then Exit Sub
'   Elabora solo le celle testuali, non quelle con formule
     On Error Resume Next
     Set WorkRange = Selection.SpecialCells _
         (xlCellTypeConstants, xlCellTypeConstants)
     For Each cell In WorkRange
         cell.Value = UCase(cell.Value)
     Next cell
End Sub
```

Potete rendere ancora più utile questa macro. Sarebbe bello, per esempio, se la macro potesse anche modificare il testo contenuto nelle celle, portandolo in minuscolo o mettendo in maiuscolo solo le iniziali delle singole parole. Un approccio consiste nel creare due macro aggiuntive: una per il minuscolo e una per le iniziali maiuscole. Un altro approccio consiste nel modificare la macro per consentirle di gestire le altre due opzioni. Se scegliete il secondo approccio, avrete bisogno di un metodo per chiedere all'utente quale tipo di intervento svolgere sulle celle.

La soluzione consiste nel visualizzare una finestra di dialogo come quella mostrata nella Figura 16.1. Potete creare questa finestra di dialogo in una UserForm nel VBE e poi la potrete visualizzare utilizzando una macro VBA. Il prossimo paragrafo offre istruzioni dettagliate per la creazione di questa finestra di dialogo.

FIGURA 16.1 Potete fare richieste all'utente utilizzando una UserForm.

PER I PIÙ CURIOSI

In VBA, il nome ufficiale di una finestra di dialogo è *UserForm*. Ma in realtà una UserForm è un oggetto che contiene la finestra di dialogo. Questa sottile distinzione non è poi così importante, quindi spesso i termini vengono usati come sinonimi.

Creare una UserForm: una panoramica

Per creare una UserForm, di solito si adottano i seguenti passaggi generali.

1. **Determinate come verrà utilizzata la finestra di dialogo e in quale punto della vostra macro VBA verrà visualizzata.**
2. **Premete Alt + F11 per attivare il VBE e inserite un nuovo oggetto UserForm.**

   Un oggetto UserForm contiene una singola UserForm.
3. **Aggiungete i controlli alla UserForm.**

   Fra i controlli disponibili vi sono elementi come le caselle di testo, i pulsanti, le caselle di controllo e le caselle di elenco.
4. **Utilizzate la finestra Proprietà per modificare le proprietà dei controlli o della stessa UserForm.**
5. **Scrivete le routine di gestione degli eventi relativi ai controlli (per esempio, una macro che venga eseguita quando l'utente fa clic su un pulsante della finestra di dialogo).**

   Queste routine vengono salvate nella finestra Codice per l'oggetto UserForm.
6. **Scrivete una routine (memorizzata in un modulo VBA) che visualizzi la finestra di dialogo all'utente.**

Non preoccupatevi se al momento alcuni di questi passaggi vi sembrano oscuri. I paragrafi seguenti forniscono ulteriori dettagli, insieme alle istruzioni dettagliate, per la creazione di una UserForm.

Durante la progettazione di una UserForm, state creando quella che gli sviluppatori chiamano *interfaccia utente grafica* (GUI) dell'applicazione.

Prendetevi del tempo per considerare quale dovrebbe essere l'aspetto della finestra e come pensate che i vostri utenti vogliano interagire con gli elementi della UserForm. Cercate di guidarli attraverso i passaggi necessari per interagire con la finestra, considerando attentamente la disposizione e la formulazione dei controlli. Come molti altri aspetti di VBA, più lo userete, più tutto vi risulterà più chiaro.

Usare le UserForm

Ogni finestra di dialogo creata viene memorizzata nel proprio oggetto UserForm, una finestra di dialogo per UserForm. Per creare e accedere a questi oggetti UserForm si usa il Visual Basic Editor.

Inserire una nuova UserForm

Potete inserire un oggetto UserForm seguendo questi passaggi.

1. **Attivate il VBE premendo Alt + F11.**
2. **Selezionate nel riquadro Progetto la cartella di lavoro che ospiterà la UserForm.**
3. **Selezionate il comando Inserisci ➪ UserForm.**

   VBE inserisce un nuovo oggetto UserForm, che contiene una finestra di dialogo vuota.

La Figura 16.2 mostra una UserForm: una finestra di dialogo vuota. Il vostro compito, se deciderete di accettarlo, è di aggiungerle alcuni controlli.

FIGURA 16.2 Un nuovo oggetto UserForm.

Aggiungere controlli a una UserForm

Quando si attiva una UserForm, il VBE visualizza la Casella degli strumenti in una finestra mobile (fate riferimento alla Figura 16.2). Per aggiungere controlli alla UserForm si utilizzano gli strumenti disponibili in questa Casella degli strumenti. Se, per qualche motivo, la Casella degli strumenti non dovesse comparire quando attivate la UserForm, selezionate Visualizza ➪ Casella degli strumenti.

Per aggiungere un elemento, fate innanzitutto clic sul controllo desiderato nella Casella degli strumenti e poi, per crearlo, trascinatelo nella finestra di dialogo. Dopo aver aggiunto un controllo, potete spostarlo e ridimensionarlo a piacere.

La Tabella 16.1 elenca i vari strumenti disponibili e le loro funzionalità. Per determinare lo strumento, posizionate il puntatore del mouse sul controllo e leggete la descrizione che appare nella casella a scomparsa.

TABELLA 16.1 Controlli disponibili della Casella degli strumenti

Controllo	Che cosa fa
Etichetta (Label)	Mostra un testo.
Casella di testo (TextBox)	Consente all'utente di inserire del testo.
Casella combinata (ComboBox)	Visualizza un elenco a discesa.
Casella di riepilogo (ListBox)	Visualizza un elenco di elementi.
Casella di controllo (CheckBox)	Fornisce opzioni di tipo On/Off o Sì/No.
Pulsante di opzione (OptionButton)	Consente all'utente di selezionare un'opzione in un gruppo di scelte.
Pulsante interruttore (ToggleButton)	Consente all'utente di attivare o disattivare un pulsante.
Cornice (Frame)	Contiene altri controlli.
Pulsante di comando (CommandButton)	Aggiunge un pulsante su cui potete fare clic.
Schede (TabStrip)	Visualizza delle schede.
Pagine (MultiPage)	Crea un contenitore a schede per altri oggetti.
Barra di scorrimento (ScrollBar)	Consente all'utente di trascinare una barra per modificare un'impostazione.
Pulsante di selezione (SpinButton)	Consente all'utente di fare clic su un pulsante per modificare un valore.
Immagine (Image)	Contiene un'immagine.
RefEdit	Consente all'utente di selezionare un intervallo.

Modificare le proprietà di un controllo della UserForm

Ogni controllo che aggiungete a una UserForm è dotato di proprietà che ne determinano l'aspetto o il comportamento. Inoltre, la stessa UserForm ha il proprio insieme di proprietà. Potete modificare queste proprietà con una delle tante finestre Proprietà. La Figura 16.3 mostra l'aspetto della finestra Proprietà per un controllo CommandButton.

FIGURA 16.3 La finestra Proprietà relativa a un controllo della UserForm.

Per visualizzare la finestra Proprietà, premete il tasto F4. In alternativa, potete selezionare Visualizza ➪ Finestra Proprietà. Le proprietà elencate nella finestra dipendono da quello che avete selezionato. Cambiando il controllo selezionato, anche le proprietà cambieranno in modo appropriato per quel controllo. Per nascondere la finestra Proprietà, fate clic sul pulsante Chiudi nella barra del titolo. Per riattivarla, premete di nuovo F4.

Fra le proprietà dei controlli trovate le seguenti:

- Name (nome);
- Width (larghezza);
- Height (altezza);
- Value (valore);
- Caption (etichetta).

Ogni controllo ha un proprio insieme di proprietà (sebbene molti controlli abbiano alcune proprietà in comune). Per modificare una proprietà utilizzando la finestra Proprietà, svolgete le seguenti operazioni.

1. **Assicuratevi che sia selezionato il controllo corretto nella UserForm.**
2. **Assicuratevi che la finestra Proprietà sia visibile (se non lo è, premete il tasto F4).**
3. **Nella finestra Proprietà, fate clic sulla proprietà che desiderate modificare.**
4. **Apportate la modifica nella parte destra della finestra Proprietà.**

Se selezionate la UserForm stessa (non un controllo della UserForm), potete utilizzare la finestra Proprietà per modificare le proprietà della UserForm.

Il Capitolo 17 descrive tutto ciò che occorre conoscere sull'utilizzo dei controlli della finestra di dialogo.

CONSIGLIO

Alcune delle proprietà della UserForm fungono da impostazioni predefinite anche per i nuovi controlli che inserirete nella UserForm stessa. Se, per esempio, modificate la proprietà Font della UserForm, i controlli che le aggiungerete utilizzeranno quello stesso carattere. I controlli già presenti sulla UserForm non saranno invece interessati dalla modifica.

Visualizzare la finestra Codice della UserForm

Ogni oggetto UserForm ha un modulo Codice che contiene il codice VBA (le routine di gestione degli eventi) che viene eseguito quando l'utente utilizza tale finestra di dialogo. Per visualizzare il modulo Codice, premete F7. Finché non aggiungerete routine, la finestra Codice sarà vuota. Premete Maiusc + F7 per tornare alla UserForm.

Ecco un altro modo per passare dalla finestra Codice alla visualizzazione della UserForm: usate i pulsanti Visualizza codice e Visualizza oggetto presenti nella barra del titolo del riquadro Progetto. Oppure fate clic destro sulla UserForm e scegliete Visualizza codice. Se state visualizzando il codice, fate doppio clic sul nome della UserForm nel riquadro Progetto e tornerete alla UserForm.

Visualizzare una UserForm

Per visualizzare una UserForm utilizzate il metodo Show dell'oggetto UserForm in una routine VBA.

RICORDA!

La macro che visualizza la finestra di dialogo deve trovarsi in un modulo VBA, e non nella finestra Codice della UserForm.

La seguente routine visualizza la finestra di dialogo denominata UserForm1:

```
Sub ShowDialogBox()
     UserForm1.Show
'   Altre istruzioni
End Sub
```

Quando Excel visualizza la finestra di dialogo, la macro ShowDialogBox si arresta fino a quando l'utente non chiuderà la finestra di dialogo. A questo punto, VBA eseguirà le istruzioni rimanenti nella routine. Il più delle volte, non sarà presente altro codice nella routine. Come vedremo in seguito, le routine di gestione degli eventi si inseriscono nella finestra Codice della UserForm. Queste routine si attivano quando l'utente lavora con i controlli della UserForm.

Utilizzare le informazioni di una UserForm

Il VBE fornisce un nome a ciascun controllo che aggiungete a una UserForm. Il nome del controllo corrisponde alla sua proprietà Name. Potete usare questo nome per far riferimento a un particolare controllo dal codice. Per esempio, se aggiungete un controllo CheckBox a una UserForm denominata UserForm1, come impostazione predefinita il controllo CheckBox si chiamerà CheckBox1. Potete utilizzare la finestra Proprietà per attivare questo controllo già attivato. Oppure potete scrivere questo codice:

```
UserForm1.CheckBox1.Value = True
```

Il più delle volte, scriverete del codice per la UserForm nel modulo Codice della UserForm. In questo caso, potete evitare di specificare il qualificatore dell'oggetto UserForm e abbreviare l'istruzione in questo modo:

```
CheckBox1.Value = True
```

Il codice VBA può anche controllare varie proprietà dei controlli e intraprendere le azioni appropriate. La seguente istruzione esegue una macro denominata PrintReport se la casella di controllo (denominata CheckBox1) è selezionata:

```
If CheckBox1.Value = True Then Call PrintReport
```

CONSIGLIO

Di solito è una buona idea cambiare il nome predefinito che il VBE assegna ai controlli, scegliendo qualcosa di più significativo. Potreste considerare di nominare la casella di controllo cbxPrintReport. Notate il prefisso di tre lettere (cbx). Non vi è alcun motivo tecnico per utilizzare il prefisso cbx. Questo è solo un esempio di impiego di una convenzione di denominazione per differenziare gli oggetti in VBA. L'utilizzo di una convenzione di denominazione come questa migliora la leggibilità del codice. Alla fine, la convenzione che sceglierete di usare è solo una questione di comodità.

Una UserForm di esempio

L'esempio di UserForm presentato in questo paragrafo è una versione avanzata della macro ChangeCase vista a inizio capitolo. La versione originale di questa macro modifica il testo nelle celle selezionate, ponendolo in maiuscole. Questa versione modificata utilizza una UserForm per chiedere all'utente quale tipo di modifica apportare: maiuscole, minuscole o iniziali maiuscole.

Questa finestra di dialogo deve chiedere all'utente un'informazione: il tipo di intervento da apportare al testo. Poiché l'utente ha tre scelte, la soluzione migliore è una finestra di dialogo con tre controlli di tipo OptionButton. La finestra di dialogo richiede anche altri due pulsanti: OK e Annulla. Facendo clic sul pulsante OK, viene eseguito il codice che esegue il lavoro; facendo clic sul pulsante Annulla, la macro termina senza fare nulla.

CONSIGLIO

Questa cartella di lavoro è disponibile, in originale inglese, sul sito web del libro. Ma datemi retta: imparerete di più seguendo i passaggi indicati e creando la routine da soli.

Creare la UserForm

I seguenti passaggi creano la UserForm. Iniziate con una cartella di lavoro vuota e seguite questi passi.

1. **Premete Alt + F11 per richiamare il VBE.**
2. **Se nel riquadro Progetto sono presenti più progetti, selezionate quello corrispondente alla cartella di lavoro che state utilizzando.**
3. **Selezionate il comando Inserisci ➪ UserForm.**

   VBE inserirà un nuovo oggetto UserForm, con una finestra di dialogo vuota.
4. **Premete F4 per visualizzare la finestra Proprietà.**
5. **Nella finestra Proprietà, modificate la proprietà Caption della finestra di dialogo e scrivete: Maiuscole e minuscole.**
6. **(Facoltativo) La finestra di dialogo è un po' troppo grande, quindi è consigliabile fare clic e utilizzare le maniglie (sul lato destro e inferiore) per renderla più piccola.**

   Potete eseguire il passaggio 6 anche dopo aver collocato nella finestra di dialogo tutti i controlli necessari.

Aggiungere i CommandButton

Pronti per aggiungere due CommandButton – OK e Annulla – alla finestra di dialogo? Procedete come segue.

1. **Assicuratevi che la Casella degli strumenti sia visualizzata; in caso contrario, selezionate Visualizza ➪ Casella degli strumenti.**
2. **Se la finestra Proprietà non è visibile, premete F4 per visualizzarla.**
3. **Nella Casella degli strumenti trascinate un Pulsante di comando nella finestra di dialogo, per crearlo.**

   Come potete vedere nella finestra Proprietà, il pulsante ha un nome (Name) e una didascalia (Caption) predefiniti: CommandButton1.
4. **Assicuratevi che il controllo CommandButton sia selezionato; quindi attivate la finestra Proprietà e modificate le seguenti proprietà:**

Proprietà	Modifica in
Name	OKButton
Caption	OK
Default	True

5. **Aggiungete alla UserForm un secondo oggetto CommandButton e modificate le seguenti proprietà:**

Proprietà	Modifica in
Name	CancelButton
Caption	Annulla
Cancel	True

6. **Regolate le dimensioni e la posizione dei controlli in modo che la finestra di dialogo sia simile a quella rappresentata nella Figura 16.4.**

FIGURA 16.4 La UserForm con due controlli CommandButton.

Aggiungere gli OptionButton

In questo paragrafo, aggiungerete alla finestra di dialogo tre pulsanti di opzione. Prima di aggiungere questi pulsanti, aggiungerete un oggetto Frame (cornice) per contenerli. Tale oggetto non è obbligatorio, ma dà alla finestra di dialogo un tocco più professionale, in modo che chi la userà non abbia l'impressione che sia stata realizzata da un dilettante.

Per aggiungere gli OptionButton, impiegate la seguente procedura.

1. **Nella Casella degli strumenti, fate clic sullo strumento Cornice e trascinatelo nella finestra di dialogo.**

   Questo passaggio crea la cornice che conterrà i pulsanti di opzioni.

2. **Utilizzate la finestra Proprietà per modificare il titolo (Caption) della cornice in Opzioni.**

3. **Nella Casella degli strumenti, fate clic sullo strumento Pulsante di opzione e trascinatelo nella finestra di dialogo (all'interno della cornice).**

   In questo modo creerete un controllo OptionButton.

4. **Selezionate il pulsante e utilizzate la sua finestra Proprietà per modificare le seguenti proprietà:**

Proprietà	Modifica in
Name	OptionUpper
Caption	Maiuscole
Accelerator	A
Value	True

   L'impostazione True della proprietà Value rende questo OptionButton il pulsante predefinito.

5. **Aggiungete un altro Pulsante di opzione e utilizzate la sua finestra Proprietà per modificare le seguenti proprietà:**

Proprietà	Modifica in
Name	OptionLower
Caption	Maiuscole
Accelerator	I

6. **Aggiungete un terzo Pulsante di opzione e utilizzate la sua finestra Proprietà per modificare le seguenti proprietà:**

Proprietà	Modifica in
Name	OptionProper
Caption	Iniziali maiuscole
Accelerator	N

7. **Regolate le dimensioni e la posizione dei pulsanti di opzioni, della cornice e della finestra di dialogo.**

   Alla fine la UserForm dovrebbe avere l'aspetto rappresentato nella Figura 16.5.

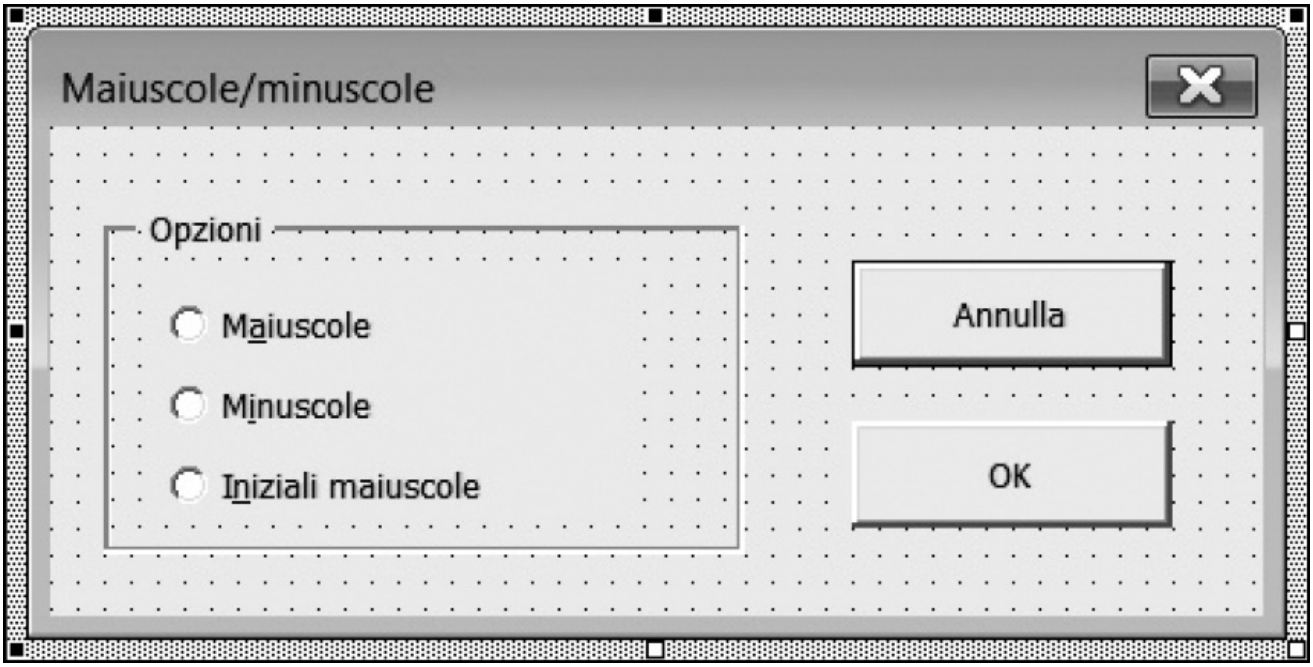

FIGURA 16.5 Questa è la UserForm dopo l'aggiunta di tre controlli Pulsante di opzione all'interno di un controllo Cornice.

Se desiderate visualizzare un'anteprima per vedere l'aspetto finale della UserForm, premete F5. Non funziona ancora nessun controllo, quindi dovrete fare clic sulla X rossa nella barra del titolo per chiuderla.

La proprietà Accelerator determina quale lettera nella didascalia è sottolineata e, cosa più importante, quale combinazione di tasti Alt + *tasto* seleziona quel controllo. Per esempio, potete selezionare l'opzione Maiuscole premendo Alt + A, perché la A è sottolineata. I tasti di scelta rapida sono opzionali, ma molti utenti preferiscono utilizzare la tastiera per usare le finestre di dialogo.

Potreste chiedervi perché i Pulsanti di opzione abbiano questi tasti di scelta rapida, mentre i Pulsanti di comando no. Generalmente, i pulsanti OK e Annulla non possiedono tasti di scelta rapida, perché sono già accessibili dalla tastiera. Per scegliere OK basta premere Invio, perché la proprietà Default del controllo è True. Per scegliere Annulla basta premere Esc, perché la proprietà Cancel del controllo è True.

Aggiungere la routine di gestione degli eventi

Ora è il momento di fare in modo che la UserForm faccia effettivamente qualcosa. Ecco come aggiungere una routine di gestione degli eventi ai pulsanti Annulla e OK.

1. **Fate doppio clic sul pulsante Annulla.**

   Il VBE attiva la finestra Codice per la UserForm e inserisce una routine vuota:

   ```
   Private Sub CancelButton_Click()
   End Sub
   ```

   La routine denominata CancelButton_Click viene eseguita quando fate clic sul pulsante Annulla, ma solo quando la finestra di dialogo viene visualizzata. In altre parole, facendo clic sul pulsante Annulla *mentre progettate* la finestra di dialogo non richiamerete la routine. Poiché la proprietà Cancel del pulsante Annulla è impostata su True, anche premendo Esc si attiva la routine CancelButton_Click.

2. **Inserite la seguente istruzione all'interno della routine (quindi prima dell'istruzione End Sub):**

   ```
   Unload UserForm1
   ```

   Questa istruzione chiude la UserForm se si fa clic sul pulsante Annulla.

3. **Premete Maiusc + F7 per tornare alla UserForm.**

4. **Fate doppio clic sul pulsante OK.**

   VBE attiva la finestra Codice per la UserForm e inserisce una routine Sub vuota denominata OKButton_Click.

   Quando la UserForm viene visualizzata, facendo clic su OK si esegue questa routine. Poiché questo pulsante ha la proprietà Default impostata su True, anche premendo Invio si esegue la routine OKButton_Click.

5. **Inserite il seguente codice:**

   ```
   Private Sub OKButton_Click()
        Dim WorkRange As Range
        Dim cell As Range

   '   Elabora le celle testuali, non quelle con formule
   ```

```
      On Error Resume Next
      Set WorkRange = Selection.SpecialCells _
          (xlCellTypeConstants, xlCellTypeConstants)

      '   Maiuscole
      If OptionUpper Then
          For Each cell In WorkRange
               cell.Value = UCase(cell.Value)
          Next cell
      End If

'   Minuscole
      If OptionLower Then
          For Each cell In WorkRange
               cell.Value = LCase(cell.Value)
          Next cell
      End If

'   Iniziali maiuscole
      If OptionProper Then
          For Each cell In WorkRange
               cell.Value = Application. _
                 WorksheetFunction.Proper(cell.Value)
          Next cell
      End If
      Unload UserForm1
End Sub
```

Il codice precedente è una versione avanzata della macro ChangeCase presentata all'inizio di questo capitolo. La macro è composta da tre blocchi distinti di codice. Questo codice utilizza tre strutture If-Then; ognuna contiene un ciclo For Each-Next. Viene eseguito un solo blocco, a seconda dell'OptionButton selezionato dall'utente. Al termine del lavoro l'ultima istruzione (Unload) chiude la finestra di dialogo.

Creare la macro per visualizzare la finestra di dialogo

Il progetto è quasi giunto al termine. L'unica cosa che manca è un modo per visualizzare la finestra di dialogo. Le seguenti istruzioni aiutano a creare la routine che consente di visualizzare la finestra di dialogo.

1. **Nella finestra di VBE, selezionate il comando Inserisci ➪ Modulo.**

   VBE aggiunge al progetto un modulo VBA vuoto (denominato Modulo1).

2. **Inseritevi il seguente codice:**

```
Sub ChangeCase()
      If TypeName(Selection) = "Range" Then
          UserForm1.Show
      Else
          MsgBox "Selezionare un intervallo.", vbCritical
      End If
End Sub
```

Questa routine è piuttosto semplice. Controlla che sia selezionato un intervallo. In tal caso, visualizza la finestra di dialogo (utilizzando il metodo Show). L'utente interagisce poi con questa finestra e viene eseguito il codice memorizzato nella finestra Codice della UserForm. Se non è selezionato alcun intervallo, all'utente viene visualizzato un MsgBox con il testo *Selezionare un intervallo*.

Rendere disponibile la macro

A questo punto, tutto dovrebbe funzionare correttamente. Ma c'è ancora bisogno di un modo semplice per richiamare la macro. Ecco i passaggi necessari per assegnare un tasto di scelta rapida (Ctrl + Shift + C) per eseguire la macro ChangeCase.

1. **Attivate la finestra di Excel premendo Alt + F11.**
2. **Selezionate il comando Sviluppo ➪ Codice ➪ Macro o premete Alt + F8.**
3. **Nella finestra di dialogo Macro, selezionate la macro ChangeCase.**
4. **Fate clic sul pulsante Opzioni.**

   Excel visualizzerà la finestra di dialogo Opzioni macro.

5. **Inserite C come tasto di scelta rapida (vedi la Figura 16.6).**
6. **Inserite una descrizione della macro nel campo Descrizione.**
7. **Fate clic OK.**
8. **Fate clic su Annulla quando si ritorna alla finestra di dialogo Macro.**

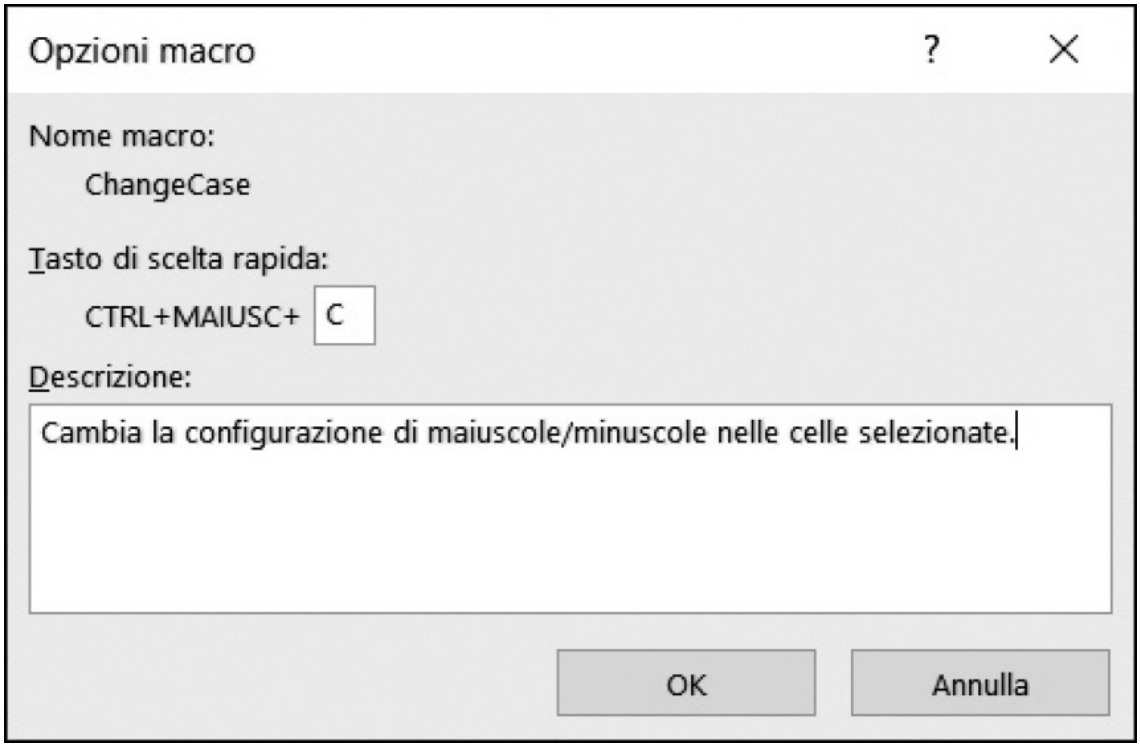

FIGURA 16.6 Assegnate il tasto di scelta rapida che richiamerà la macro ChangeCase.

Dopo aver eseguito questa operazione, premendo Ctrl + Maiusc + C eseguirete la macro ChangeCase, che, se in quel momento è selezionato un intervallo, visualizzerà la UserForm.

Potete rendere disponibile questa macro anche nella barra di accesso rapido. Fate clic destro sulla barra di accesso rapido e selezionate dal menu contestuale il comando Personalizza barra di accesso rapido. Verrà visualizzata la finestra di dialogo Opzioni di Excel e troverete la macro ChangeCase elencata sotto Macro (vedi la Figura 16.7).

Provare la macro

Giunti a questo punto occorre provare la macro e la finestra di dialogo, per assicurarsi che funzionino correttamente.

1. **Attivate un foglio di lavoro (qualsiasi foglio di lavoro in qualsiasi cartella di lavoro).**
2. **Selezionate alcune celle contenenti del testo.**

   Potete anche selezionare intere righe o colonne.
3. **Premete la combinazione di tasti Ctrl + Maiusc + C.**

   Verrà aperta la UserForm. La Figura 16.8 mostra il suo aspetto.

FIGURA 16.7 Aggiunta della macro ChangeChase alla barra di accesso rapido.

4. **Fate la vostra scelta e poi fate clic su OK.**

   Se avete svolto tutto correttamente, la macro apporterà la modifica specificata al testo contenuto nelle celle selezionate.

FIGURA 16.8 La UserForm in azione.

Se provate questa routine quando è selezionata una sola cella, verranno elaborate tutte le celle del foglio di lavoro. Questo comportamento è un sottoprodotto dell'uso del metodo SpecialCells. Se preferite poter elaborare anche una sola cella, modificate il primo blocco di codice come segue:

```
If Selection.Count = 1 Then
     Set WorkRange = Selection
Else
     Set WorkRange = Selection.SpecialCells _
         (xlCellTypeConstants, xlCellTypeConstants)
End If
```

La Figura 16.9 mostra il foglio di lavoro dopo la conversione del testo in lettere maiuscole. Notate che la formula contenuta nella cella B16 e la data contenuta nella cella B17 non sono state modificate. Come previsto, la macro

funziona solo sulle celle contenenti del testo.

	A	B	C	D
1				
2		GENNAIO		
3		FEBBRAIO		
4		MARZO		
5		APRILE		
6		MAGGIO		
7		GIUGNO		
8		LUGLIO		
9		AGOSTO		
10		SETTEMBRE		
11		OTTOBRE		
12		NOVEMBRE		
13		DICEMBRE		
14				
15				
16	Formula:	martedì 14 maggio 2019		
17	Data:	sabato 14 maggio 2022		
18				

FIGURA 16.9 La macro ha convertito il testo in lettere maiuscole.

Finché la cartella di lavoro è aperta, potrete eseguire la macro da qualsiasi altra cartella di lavoro. Chiudendo la cartella di lavoro che contiene la vostra macro, la combinazione di tasti Ctrl + Maiusc + C non produrrà più alcun risultato.

Se la macro non funziona correttamente, ricontrollate i passaggi precedenti per individuare e correggere l'errore. E… non vi allarmate: eliminare gli errori (il famoso debugging) è un'attività del tutto normale nello sviluppo di macro. Come ultima risorsa, provate a scaricare la cartella di lavoro completa (in originale inglese) dal sito web di questo libro e provate a capire che cosa è andato storto.

Capitolo 17

Utilizzare i controlli per le UserForm

IN QUESTO CAPITOLO

- **I tipi di controlli della finestra di dialogo**
- **Modificare le proprietà dei vari controlli**
- **Manipolare i controlli della finestra di dialogo nell'oggetto UserForm**

Un utente interagisce con una finestra di dialogo personalizzata (nota anche con il nome di *UserForm*) utilizzando vari tipi di controlli (pulsanti, caselle di modifica, pulsanti di opzione e così via) disposti in modo opportuno nella finestra di dialogo. Il vostro codice VBA utilizza poi le sue risposte per determinare quali azioni intraprendere. Avete a disposizione molti controlli, che sono l'argomento di questo capitolo.

Se avete provato l'esempio pratico del Capitolo 16, avrete già acquisito una certa esperienza con i controlli per le UserForm. Questo capitolo colma le lacune rimanenti.

I controlli per le finestre di dialogo

In questo paragrafo, impareremo ad aggiungere controlli a una UserForm, ad assegnare loro nomi significativi e a modificare le loro proprietà.

RICORDA!

Prima di poter eseguire queste operazioni, è necessario disporre di una UserForm, che si ottiene scegliendo dal VBE il comando Inserisci ➪ UserForm. Quando aggiungete una UserForm, assicuratevi che nel riquadro Progetto sia selezionato il progetto corretto (nel caso sia disponibile più di un progetto).

Aggiungere controlli

Stranamente, il VBE non ha un menu di comandi che consentano di aggiungere controlli a una finestra di dialogo. Per aggiungere controlli si passa dalla Casella degli strumenti, descritta nel Capitolo 16. Quando si attiva una UserForm nel VBE, in genere la Casella degli strumenti viene visualizzata automaticamente. In caso contrario, potete richiamarla selezionando il comando Visualizza ➪ Casella degli strumenti.

Per aggiungere un controllo alla UserForm procedete come segue.

1. **Fate clic su uno strumento nella Casella degli strumenti corrispondente al controllo che desiderate aggiungere.**
2. **Fate clic nella UserForm ed eseguite trascinamenti per ridimensionare e posizionare il controllo.**

   In alternativa, potete semplicemente trascinare un controllo dalla Casella degli strumenti alla UserForm per creare un controllo di dimensioni predefinite. La Figura 17.1 mostra una UserForm contenente alcuni controlli: due Pulsanti di opzioni (OptionButtons) all'interno di una Cornice (Frame), una Casella combinata (ComboBox), una Casella di controllo (CheckBox), una Barra di scorrimento (ScrollBar) e un Pulsante di comando (CommandButton).

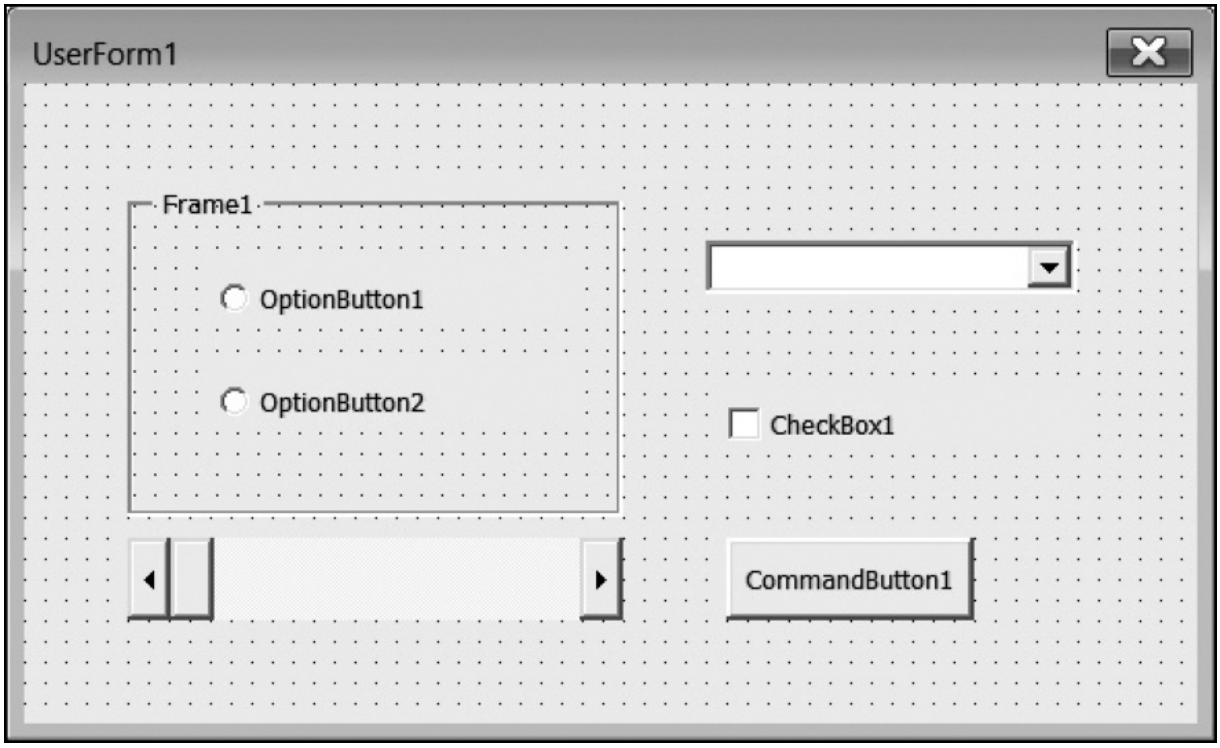

FIGURA 17.1 Una UserForm nel VBE, con alcuni controlli.

CONSIGLIO

Una UserForm può contenere una griglia verticale e orizzontale, che aiuta ad allineare i controlli aggiunti. Quando aggiungete o spostate un controllo, questo si *aggancia* alla griglia. Se non gradite questa funzione, potete disattivarla procedendo nel seguente modo.

1. **Scegliete nel VBE il comando Strumenti ➪ Opzioni.**
2. **Nella finestra di dialogo Opzioni, selezionate la scheda Generale.**
3. **Impostate le opzioni desiderate nella sezione Impostazioni griglia form.**

Le proprietà dei controlli

Ogni controllo che si aggiunge a una UserForm ha specifiche proprietà che ne determinano l'aspetto e il comportamento. Potete modificare le proprietà di un controllo nei due casi seguenti.

» In fase di progettazione, quando si progetta la UserForm. Si procede manualmente, usando la finestra Proprietà.

» *Runtime*, ovvero quando la macro è in esecuzione. Scrivendo il codice VBA. Le modifiche apportate *runtime* sono sempre temporanee; valgono per la copia della finestra di dialogo che state visualizzando, non per l'oggetto UserForm che avete progettato.

Quando si aggiunge un controllo a una UserForm, è quasi sempre necessario apportare alcune modifiche in fase di progettazione alle sue proprietà. Per apportare queste modifiche si usa la finestra Proprietà (per visualizzare questa finestra, premete il tasto F4). La Figura 17.2 mostra la finestra Proprietà, che elenca le proprietà dell'oggetto selezionato nella UserForm: un controllo CheckBox.

FIGURA 17.2 Uso della finestra Proprietà per apportare modifiche in fase di progettazione alle proprietà di un controllo.

RICORDA!

Per modificare le proprietà di un controllo durante l'esecuzione, è necessario scrivere del codice VBA. Per esempio, potreste voler nascondere un particolare controllo quando l'utente fa clic su una casella di controllo. In tal caso, dovrete scrivere del codice per modificare la proprietà Visible di quel controllo.

Ogni controllo ha il proprio insieme di proprietà. Tutti i controlli, tuttavia, hanno alcune proprietà in comune, come Name (nome) e Width e Height (larghezza e altezza). La Tabella 17.1 elenca alcune delle proprietà comuni disponibili in molti controlli.

TABELLA 17.1 Proprietà più comuni dei controlli

Proprietà	Che cosa fa
Accelerator	La lettera sottolineata nell'etichetta del controllo. L'utente può premere la combinazione di tasti Alt + *questo tasto* per selezionare il controllo.
AutoSize	Se è True, il controllo si ridimensiona automaticamente in base al testo dell'etichetta.
BackColor	Il colore di sfondo del controllo.
BackStyle	Lo stile dello sfondo (trasparente o opaco).
Caption	Il testo che appare sul controllo.
Left e Top	Determinano la posizione del controllo.
Name	Il nome del controllo. Per impostazione predefinita, il nome di un controllo si basa sul tipo di controllo. Potete cambiare il nome scegliendo qualsiasi nome valido, ma il nome di ciascun controllo deve essere univoco all'interno della finestra di dialogo.
Picture	Un'immagine da visualizzare. L'immagine può provenire da un file oppure potete selezionare la proprietà Picture e incollare un'immagine precedentemente copiata negli Appunti.
Value	Il valore del controllo.
Visible	Se è False, il controllo è nascosto.
Width e Height	Valori che determinano rispettivamente la larghezza e l'altezza del controllo.

Quando selezionate un controllo, la finestra Proprietà mostra le sue proprietà. Per modificare una proprietà, selezionatela nella finestra Proprietà e apportate direttamente la modifica. Alcune proprietà forniscono anche un aiuto. Per esempio, se dovete modificare la proprietà TextAlign, la finestra Proprietà visualizza un elenco a discesa che contiene tutti i valori validi per la proprietà, come illustrato nella Figura 17.3.

I controlli della finestra di dialogo: tutti i dettagli

I seguenti paragrafi illustrano uno per uno i tipi di controlli che potete utilizzare nelle finestre di dialogo

personalizzate e alcune delle loro proprietà più interessanti. Non includo tutte le proprietà di ogni controllo perché ciò richiederebbe un libro quattro volte più grande (e molto, molto noioso).

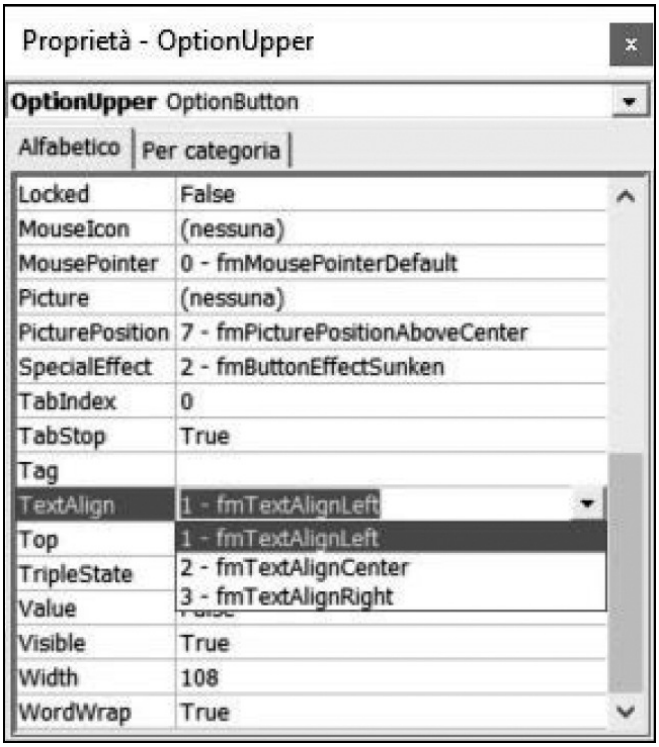

FIGURA 17.3 Potete modificare alcune proprietà selezionandole da un elenco a discesa di valori validi.

La Guida per i controlli e le proprietà è completa. Per trovare tutti i dettagli di una particolare proprietà, selezionatela nella finestra Proprietà e premete F1.

CONSIGLIO

Tutti gli esempi di questo paragrafo sono disponibili (in originale inglese) sul sito web di questo libro.

Il controllo CheckBox (Casella di controllo)

Un controllo CheckBox è utile quando occorre ottenere una scelta binaria: un sì o un no, vero o falso, acceso o spento e così via. La Figura 17.4 mostra alcuni esempi di controlli CheckBox.

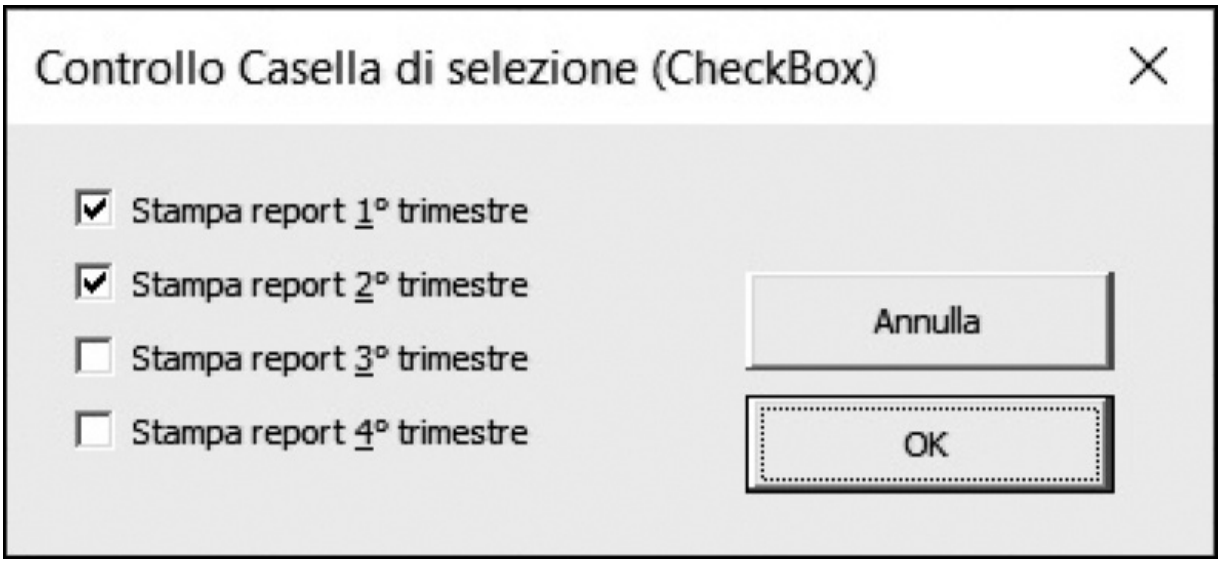

FIGURA 17.4 Controlli CheckBox di una UserForm.

Quelle che seguono sono le proprietà più utili del controllo CheckBox.

- **Accelerator:** un carattere che consente all'utente di commutare il controllo utilizzando la tastiera. Per esempio, se l'acceleratore è A, premendo Alt + A potrà commutare il valore del controllo CheckBox (da selezionato a non selezionato o viceversa). Nell'esempio rappresentato nella Figura 17.4, gli "acceleratori" sono i numeri (Alt + 1, Alt + 2 e così via).
- **ControlSource:** l'indirizzo di una cella del foglio di lavoro collegata alla CheckBox. La cella visualizza TRUE se il controllo è selezionato e FALSE se non è selezionato. È opzionale. Il più delle volte, una CheckBox non è collegata a una cella.
- **Value:** se è True, la CheckBox ha un segno di spunta. Se è False, la CheckBox non ha un segno di spunta.

WARNING

Non confondere i controlli CheckBox (la Casella di selezione) con i controlli OptionButton (il Pulsante di opzione). Sembrano simili, ma vengono usati per scopi diversi.

Il controllo ComboBox (Casella combinata)

Un controllo ComboBox è simile a un controllo ListBox (descritto più avanti nel paragrafo "Il controllo ListBox (Casella di riepilogo)"). Una ComboBox, tuttavia, è un controllo a discesa. Un'altra differenza è che l'utente può essere autorizzato a inserire un valore che non appare nell'elenco degli oggetti. La Figura 17.5 mostra due controlli ComboBox. Viene utilizzato il controllo a destra (per l'anno), per visualizzare l'elenco delle opzioni.

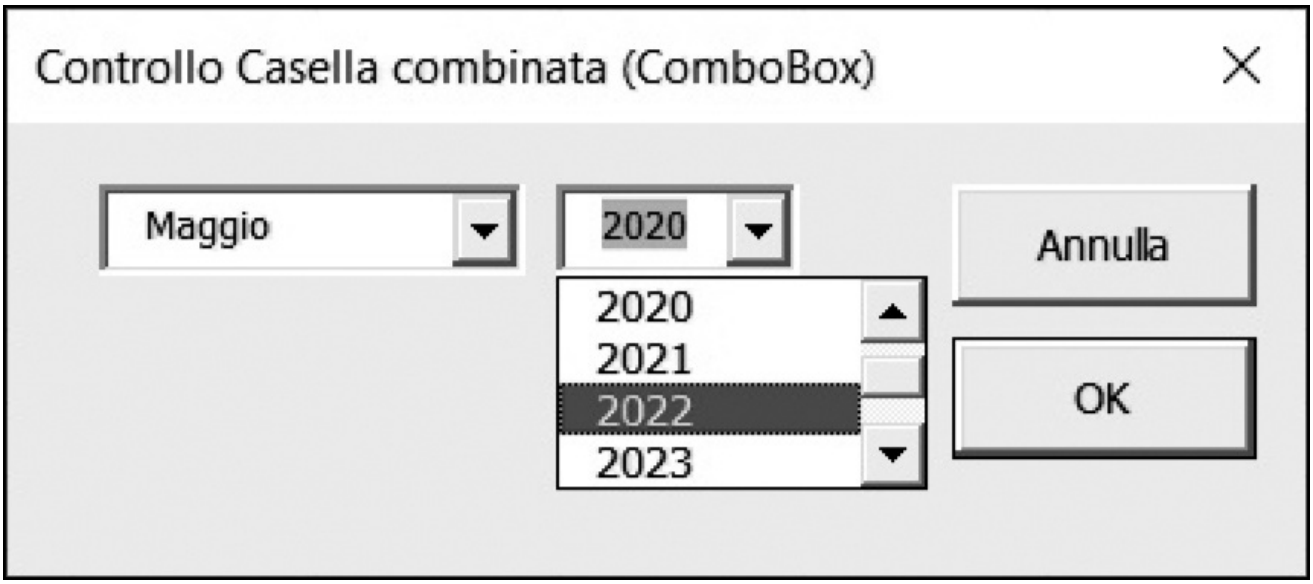

FIGURA 17.5 I controlli ComboBox nella UserForm.

Quelle che seguono sono alcune proprietà utili di controllo di ComboBox.

- **ControlSource:** una cella che memorizza il valore selezionato nel ComboBox.
- **ListRows:** numero di elementi da visualizzare quando viene aperto l'elenco.
- **ListStyle:** aspetto degli elementi presenti nell'elenco prodotto.
- **RowSource:** l'indirizzo dell'intervallo di celle che contiene l'elenco degli elementi visualizzati nel controllo ComboBox.
- **Style:** determina se il controllo funziona come un elenco a discesa o come un controllo ComboBox. Un elenco a discesa non consente all'utente di immettere un nuovo valore.
- **Value:** il testo dell'elemento selezionato dall'utente e visualizzato nella ComboBox.

PER I PIÙ CURIOSI

Se l'elenco di elementi non si trova in un foglio di lavoro, potete aggiungere gli elementi di un controllo ComboBox utilizzando il metodo AddItem. Ulteriori informazioni su questo metodo si trovano nel Capitolo 18.

Il controllo CommandButton (Pulsante di comando)

CommandButton è solo un comune pulsante su cui potete fare clic. Da solo è inutile, a meno che non si fornisca una routine di gestione degli eventi da eseguire dopo il clic sul pulsante. La Figura 17.6 mostra una finestra di dialogo con nove CommandButton. Due di questi pulsanti presentano immagini ClipArt, che vengono aggiunte copiando la ClipArt e incollandola nel campo Picture della finestra Proprietà.

FIGURA 17.6 I controlli CommandButton.

Quando fate clic su un CommandButton, viene eseguita una routine di gestione degli eventi il cui nome è composto dal nome del CommandButton, un carattere di sottolineatura e la parola `Click`. Per esempio, se il pulsante di comando si chiama IlPulsante, facendo clic su di esso viene eseguita la macro denominata IlPulsante_Click. Questa macro è memorizzata nella finestra Codice della UserForm.

Quelle elencate di seguito sono alcune proprietà utili di controllo CommandButton.

- **Cancel:** se è True, premendo Esc si esegue la macro collegata al pulsante. Solo uno dei pulsanti del modulo può avere questa opzione impostata su True.
- **Default:** se è True, premendo Invio si esegue la macro collegata al pulsante. Anche in questo caso solo uno dei pulsanti del modulo può avere questa opzione impostata su True.

Il controllo Frame (Cornice)

Un controllo Frame è una cornice che racchiude altri controlli. Si usa per scopi estetici o per raggruppare logicamente un insieme di controlli. Un Frame è particolarmente utile quando la finestra di dialogo contiene più gruppi di controlli OptionButton (vedere “Il controllo OptionButton (Pulsante di opzione)”, più avanti in questo capitolo).

Il seguente elenco descrive alcune proprietà utili di controllo del frame.

- **BorderStyle:** definisce l’aspetto della cornice.
- **Caption:** il testo visualizzato nella parte superiore della cornice. La Caption può anche essere una stringa vuota, se non desiderate che il controllo abbia un titolo.

Il controllo Image (Immagine)

Un controllo Image visualizza un’immagine. Potreste volerlo usare per visualizzare il logo della vostra azienda nella finestra di dialogo. La Figura 17.7 mostra una finestra di dialogo con un controllo Image con la foto di un micio.

Il seguente elenco descrive le proprietà più utili del controllo Image.

- **Picture:** l’immagine visualizzata.
- **PictureSizeMode:** modalità di visualizzazione dell’immagine, se la dimensione del controllo non coincide con le dimensioni dell’immagine.

FIGURA 17.7 Un controllo Image mostra una foto.

Quando fate clic sulla proprietà Picture, vi viene richiesto il nome del file. Tale immagine viene poi archiviata nella cartella di lavoro. In questo modo, se fornirete la cartella di lavoro a qualcun altro, non dovrete includere anche una copia del file grafico.

CONSIGLIO

Invece di caricare l'immagine da un file, potete copiarla e incollarla. Per cercare sul Web le immagini, attivate Excel e selezionate il comando Inserisci ➪ Illustrazioni ➪ Immagini online e cercate un'immagine da inserire nel foglio di lavoro. Quindi selezionate l'immagine e premete Ctrl + C, per copiarla negli Appunti. Attivate la UserForm, fate clic sul controllo Image e selezionate la proprietà Picture nella finestra Proprietà. Poi premete Ctrl + V per incollare l'immagine copiata. Ora potete anche eliminare l'immagine dal foglio di lavoro.

WARNING

Alcune immagini sono molto grandi e possono aumentare notevolmente le dimensioni della cartella di lavoro. Per ottenere risultati ottimali, utilizzate solo immagini compatte.

Il controllo Label (Etichetta)

Un controllo Label visualizza semplicemente del testo nella finestra di dialogo. La Figura 17.8 mostra alcuni controlli Label. Come potete vedere, avete un grande controllo sulla formattazione di un controllo Label.

FIGURA 17.8 I controlli Label possono assumere l'aspetto desiderato.

Il controllo ListBox (Casella di riepilogo)

Il controllo ListBox presenta un elenco di elementi dal quale l'utente può sceglierne uno o anche più d'uno. La Figura 17.9 mostra una finestra di dialogo contenente due controlli ListBox.

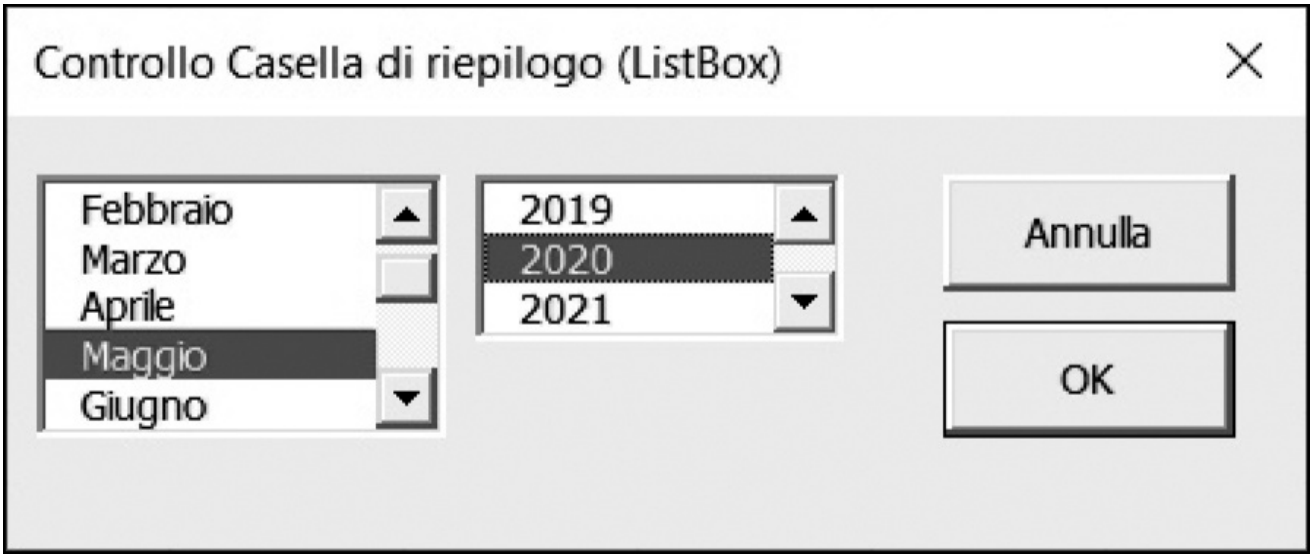

FIGURA 17.9 Controlli ListBox.

I controlli ListBox sono molto flessibili. Per esempio, potete specificare un intervallo in un foglio di lavoro che contiene gli oggetti da inserire nella ListBox e l'intervallo può essere costituito anche da più colonne. Oppure potete riempire la ListBox con oggetti predisposti tramite il codice VBA.

Se una ListBox non è grande a sufficienza per visualizzare tutti gli elementi dell'elenco, viene visualizzata anche una barra di scorrimento, in modo che l'utente possa scorrerne il contenuto per visualizzare più elementi.

Il seguente elenco descrive le proprietà più utili del controllo ListBox.

- **ControlSource:** una cella che memorizza il valore selezionato in ListBox.
- **IntegralHeight:** se è True, quando l'elenco scorre verticalmente, l'altezza della ListBox si regola automaticamente per visualizzare righe di testo complete. Se è False, quando viene fatta scorrere verticalmente, la ListBox potrebbe visualizzare righe di testo troncate. Notate che quando questa proprietà è True, al momento della visualizzazione della UserForm l'altezza effettiva della ListBox può essere leggermente diversa rispetto alla vostra impostazione originaria. In altre parole, l'altezza può essere modificata dal sistema per garantire che l'ultima voce sia interamente visibile.

» **ListStyle:** l'aspetto degli elementi dell'elenco.
» **MultiSelect:** determina se l'utente può selezionare più elementi tratti dall'elenco.
» **RowSource:** l'intervallo che contiene l'elenco degli elementi visualizzati nella ListBox.
» **Value:** il testo dell'elemento selezionato nella ListBox.

RICORDA!

Se la proprietà MultiSelect della ListBox è impostata su 1 o 2, l'utente può selezionare più elementi. In tal caso, non potete specificare una ControlSource; è necessario scrivere una macro che determini quali elementi sono selezionati. Il Capitolo 18 mostra come farlo.

Il controllo MultiPage (Pagine)

Un controllo MultiPage consente di creare finestre di dialogo con schede, come la finestra di dialogo Formato celle (quella visualizzata quando premete Ctrl + 1 in Excel). La Figura 17.10 mostra un esempio di una finestra di dialogo personalizzata che utilizza un controllo MultiPage. Questo particolare controllo offre tre pagine o schede.

Di seguito sono riportate le proprietà più utili del controllo MultiPage.

» **Style:** determina l'aspetto del controllo. Le schede possono essere visualizzate normalmente (in alto), a sinistra, come pulsanti o nascoste (nessuna scheda: è il codice VBA a determinare quale pagina visualizzare).
» **Value:** determina quale pagina o scheda è visualizzata. Il valore 0 visualizza la prima pagina, il valore 1 visualizza la seconda pagina e così via.

CONSIGLIO

Per impostazione predefinita, un controllo MultiPage ha due pagine. Per aggiungere altre pagine, fate clic destro su una scheda e selezionate Nuova pagina dal menu contestuale.

Il controllo OptionButton (Pulsante di opzione)

Gli OptionButton sono utili quando l'utente deve scegliere tra un numero limitato di elementi che si escludono a vicenda. Gli OptionButton sono sempre utilizzati in gruppi di almeno due elementi. La Figura 17.11 mostra due serie di OptionButton: ogni serie è contenuta in un Frame.

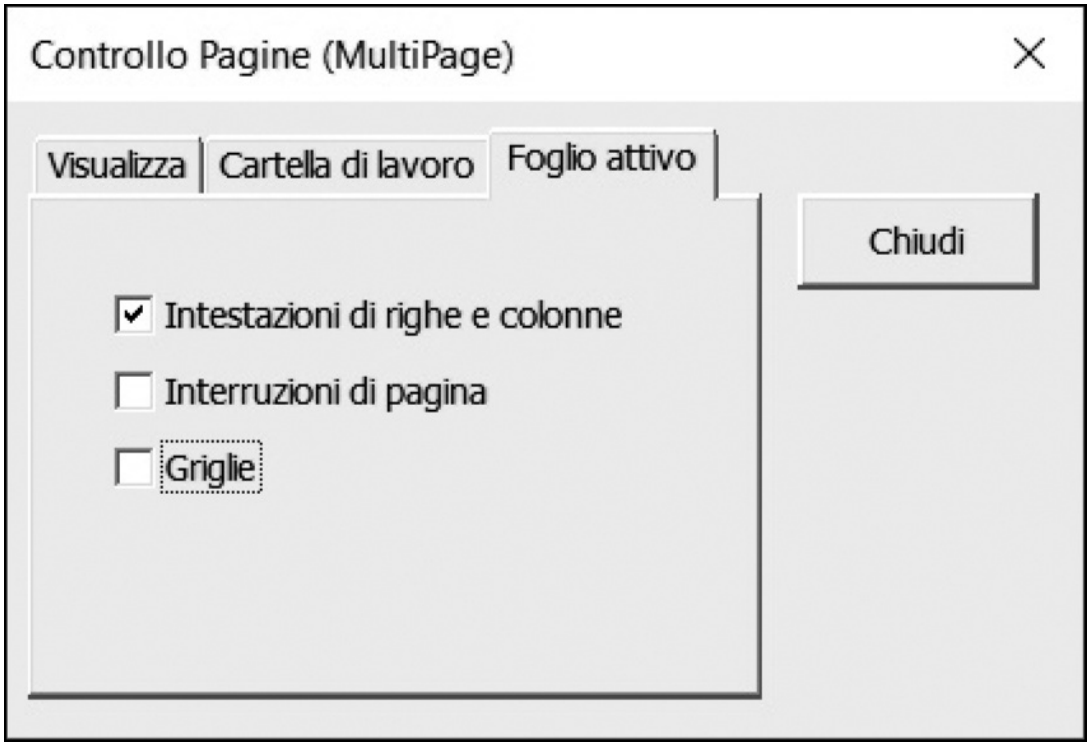

FIGURA 17.10 Un controllo MultiPage permette di creare una finestra di dialogo con schede.

FIGURA 17.11 Due serie di controlli OptionButton, ciascuno contenuto in un controllo Frame.

Di seguito è riportata una descrizione delle principali proprietà del controllo OptionButton.

- **Accelerator:** una lettera che consente all'utente di selezionare l'opzione utilizzando la tastiera. Per esempio, se l'acceleratore per un pulsante di opzione è C, premendo Alt + C potrà selezionare il controllo.
- **GroupName:** un nome che stabilisce che il pulsante di opzione è associato ad altri pulsanti di opzione, aventi la stessa proprietà GroupName.
- **ControlSource:** la cella del foglio di lavoro collegata al pulsante di opzione. La cella visualizza TRUE se il controllo è selezionato e FALSE se il controllo non è selezionato.
- **Value:** se è True, l'OptionButton è selezionato. Se è False, l'OptionButton non è selezionato.

RICORDA!

Se la finestra di dialogo contiene più di una serie di OptionButton, dovete modificare la proprietà GroupName per tutti gli OptionButton di una determinata serie. Altrimenti, tutti gli OptionButton faranno parte della stessa serie. In alternativa, potete racchiudere ciascuna serie di OptionButton in un controllo Frame, che raggruppa automaticamente gli OptionButton che contiene.

Il controllo RefEdit

In alcuni casi, potrebbe essere necessario che l'utente selezioni un intervallo di celle per passare al codice VBA un intervallo valido. Le vostre routine potranno poi utilizzare l'intervallo selezionato per le attività desiderate. Questo controllo viene utilizzato quando è necessario consentire all'utente di selezionare un intervallo in un foglio di lavoro. La Figura 17.12 mostra una UserForm con due controlli Ref Edit. La sua proprietà Value contiene l'indirizzo dell'intervallo selezionato (sotto forma di stringa di testo).

WARNING

Il controllo RefEdit a volte causa problemi nelle UserForm più complesse. Per risultati ottimali, evitate di posizionare un controllo RefEdit all'interno di un controllo Frame o MultiPage.

FIGURA 17.12 Due controlli RefEdit.

Il controllo ScrollBar (Barra di scorrimento)

Potete aggiungere un controllo ScrollBar orizzontale o verticale. Il controllo ScrollBar è simile a un controllo SpinButton (descritto più avanti). La differenza è che l'utente può trascinare il pulsante ScrollBar per modificare il valore del controllo con incrementi maggiori. Un'altra differenza è che quando fate clic sul pulsante in una barra di scorrimento verticale, il valore diminuisce, cosa leggermente controintuitiva. Quindi una ScrollBar non è sempre un buon sostituto di uno SpinButton.

La Figura 17.13 mostra un controllo ScrollBar con un orientamento orizzontale. Sotto il controllo ScrollBar in un controllo Label viene visualizzata la sua proprietà Value.

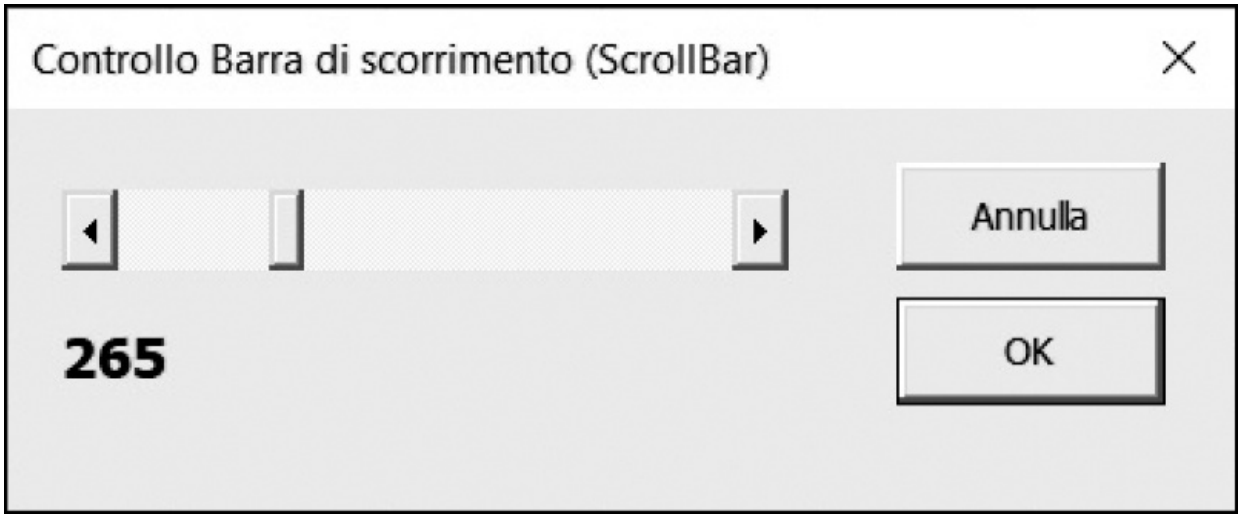

FIGURA 17.13 Un controllo ScrollBar con un controllo Label sotto di esso.

Quelle che seguono sono le proprietà più utili di un controllo ScrollBar.

- **Value:** il valore corrente del controllo.
- **Min:** il valore minimo del controllo.
- **Max:** il valore massimo del controllo.
- **ControlSource:** la cella del foglio di lavoro che visualizza il valore del controllo.
- **SmallChange:** l'estensione in base alla quale il valore del controllo viene modificato da un clic sulla barra.
- **LargeChange:** l'estensione in base alla quale il valore del controllo viene modificato facendo clic sui pulsanti laterali.

Il controllo ScrollBar è molto utile per specificare un valore che si può estendere su un'ampia gamma di valori possibili.

Il controllo SpinButton (Pulsante di selezione)

Il controllo SpinButton consente all'utente di selezionare un valore facendo clic sul controllo, che ha due frecce (una per aumentare il valore e l'altra per diminuirlo). Come un controllo ScrollBar, uno SpinButton può essere orientato orizzontalmente o verticalmente. La Figura 17.14 mostra una finestra di dialogo che utilizza due controlli SpinButton orientati verticalmente. Ogni controllo è stato collegato al controllo dalla Label sulla destra (utilizzando routine VBA).

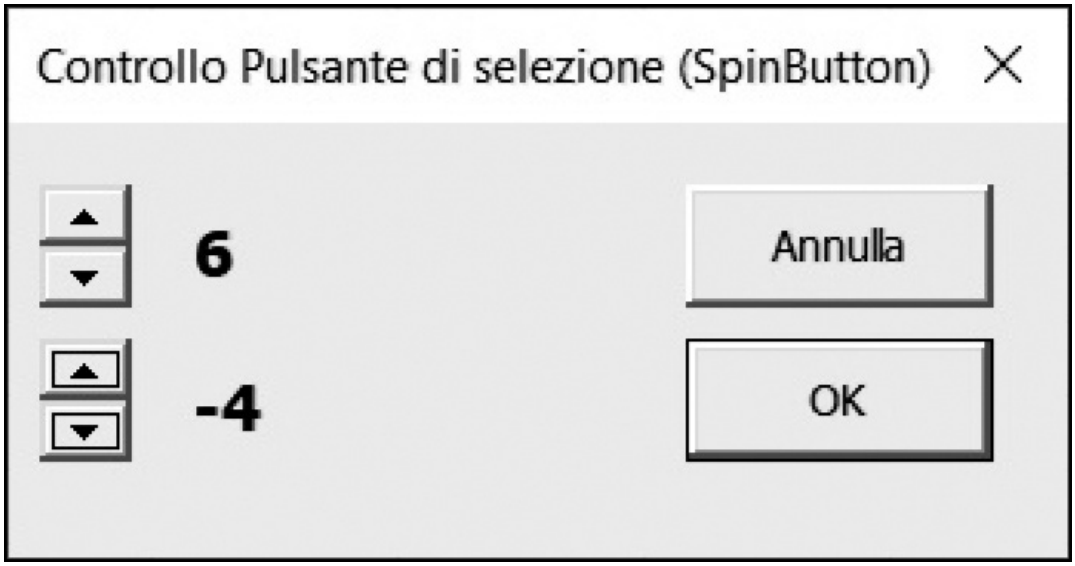

FIGURA 17.14 Controlli SpinButton.

Quelle che seguono sono le proprietà più utili di un controllo SpinButton.

- **Value:** il valore corrente del controllo.
- **Min:** il valore minimo del controllo.
- **Max:** il valore massimo del controllo.
- **ControlSource:** la cella del foglio di lavoro che visualizza il valore del controllo.
- **SmallChange:** l'importo in base al quale il valore del controllo viene modificato da un clic. Di solito, questa proprietà è impostata su 1, ma può essere qualsiasi altro valore.

WARNING

Se utilizzate ControlSource per uno SpinButton, dovete considerare che il foglio di lavoro viene ricalcolato ogni volta che viene modificato il valore del controllo. Pertanto, se l'utente dovesse modificare il valore da 0 a 12, il foglio di lavoro verrebbe ricalcolato dodici volte. Se il foglio di lavoro richiede molto tempo per il calcolo, potete evitare di utilizzare ControlSource per memorizzare il valore.

Il controllo TabStrip (Schede)

Un controllo TabStrip è progettato per esplorare diversi insiemi di valori utilizzando una singola pagina con lo stesso gruppo di controlli. Questo controllo viene utilizzato raramente, poiché il controllo MultiPage offre un modo molto più semplice per ottenere la stessa funzionalità. Potete ignorarlo e utilizzare invece il controllo MultiPage.

Il controllo TextBox (Casella di testo)

Un controllo TextBox consente all'utente di inserire del testo. La Figura 17.15 mostra una finestra di dialogo con due controlli TextBox.

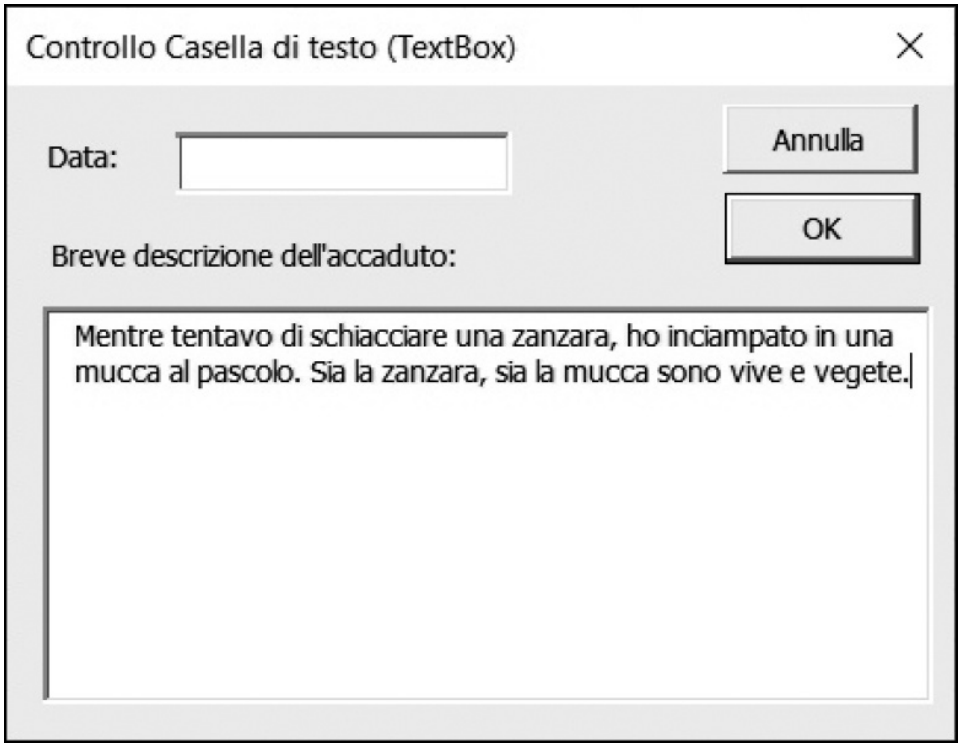

FIGURA 17.15 I controlli TextBox.

Quelle che seguono sono le proprietà più utili del controllo TextBox:

- **AutoSize:** se è True, il controllo regola automaticamente le sue dimensioni, a seconda della quantità di testo.
- **ControlSource:** l'indirizzo di una cella che contiene il testo nel controllo TextBox.
- **IntegralHeight:** se è True, quando l'elenco scorre verticalmente, l'altezza della TextBox si regola automaticamente per visualizzare righe complete di testo. Se è False, quando il contenuto della TextBox viene fatto scorrere verticalmente potrebbe visualizzare righe di testo parziali.
- **MaxLength:** il numero massimo di caratteri consentito nel controllo TextBox. Se è uguale a 0, il numero di caratteri è illimitato.
- **MultiLine:** se è True, la TextBox può visualizzare più di una riga di testo.
- **TextAlign:** determina l'allineamento del testo nella TextBox.
- **WordWrap:** determina se il controllo consente l'a capo automatico.
- **ScrollBar:** determina il tipo di barre di scorrimento per il controllo: orizzontale, verticale, entrambe o nessuna.

CONSIGLIO

Quando si aggiunge un controllo TextBox, la sua proprietà WordWrap è impostata su True e la proprietà MultiLine è impostata su False. L'effetto finale? L'a capo non funziona! Quindi, se desiderate che le parole vengano mandate a capo in un controllo TextBox, assicuratevi di impostare la proprietà MultiLine su True.

Il controllo ToggleButton (Pulsante interruttore)

Un controllo ToggleButton ha due stati: on e off. Facendo clic sul pulsante si alternano questi due stati e il pulsante cambia aspetto. Il suo valore è True (premuto) o False (non premuto). La Figura 17.16 mostra una finestra di dialogo con quattro controlli ToggleButton. Quello superiore è premuto.

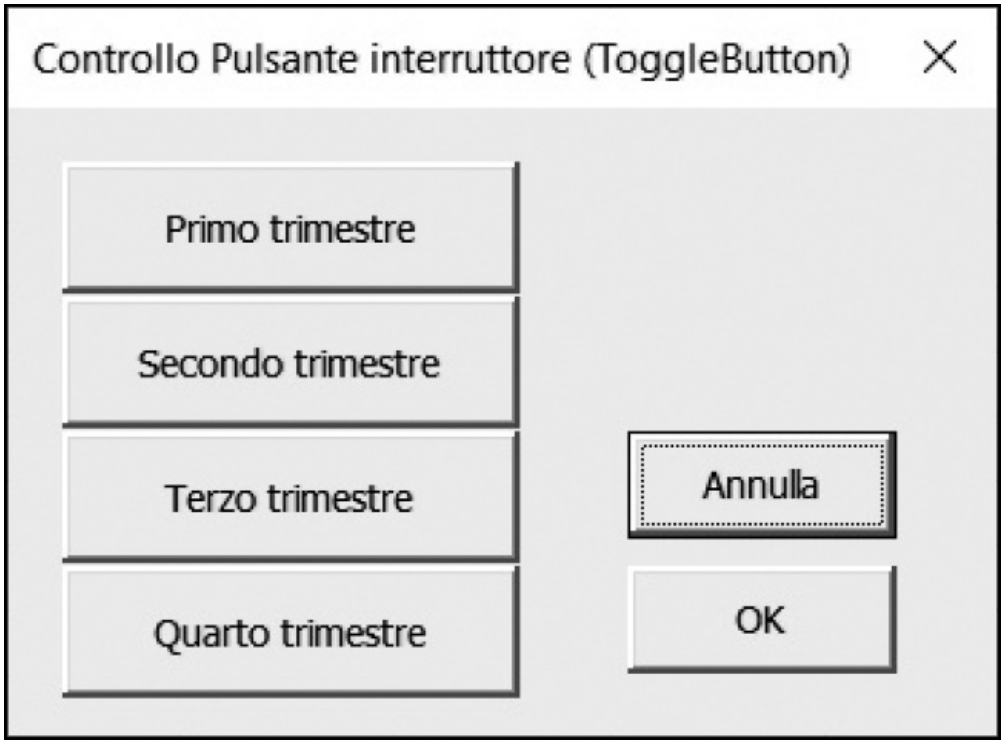

FIGURA 17.16 I controlli ToggleButton.

Gestire i controlli nella finestra di dialogo

In questo paragrafo presento alcuni suggerimenti utili per spostare, ridimensionare e allineare i controlli, per farli apparire correttamente nelle UserForm.

Spostare e ridimensionare i controlli

Dopo aver posizionato un controllo in una finestra di dialogo, potete spostarlo e ridimensionarlo usando le tecniche standard del mouse. Oppure, per un controllo preciso, potete utilizzare la finestra Proprietà per immettere un valore per la proprietà Height, Width, Left o Top del controllo.

CONSIGLIO

Potete selezionare più controlli premendo Ctrl + clic su di essi. Oppure potete selezionare con un'area un gruppo di controlli. Quando sono selezionati più controlli, la finestra Proprietà visualizza solo le proprietà comuni a tutti i controlli selezionati. Potete modificare queste proprietà comuni e la modifica verrà apportata a tutti i controlli selezionati, il che è molto più rapido rispetto a modificarli uno per volta.

Un controllo può nascondere un altro controllo; in altre parole, potete impilare un controllo sopra un altro. A meno che non abbiate una buona ragione per farlo, assicuratevi di non aver sovrapposto più controlli.

Allineare e distribuire i controlli

Il menu Formato della finestra di VBE fornisce diversi comandi per aiutarvi ad allineare e distribuire con precisione i controlli in una finestra di dialogo. Prima di utilizzare questi comandi, selezionate i controlli su cui desiderate lavorare. Questi comandi sono abbastanza auto-esplicativi.

La Figura 17.17 mostra una finestra di dialogo con diversi controlli CheckBox che stanno per essere allineati.

FIGURA 17.17 Scegliete Formato ➪ Allinea per modificare l'allineamento dei controlli nella UserForm.

CONSIGLIO

Quando selezionate più controlli, l'ultimo controllo selezionato viene visualizzato con maniglie bianche anziché con le normali maniglie nere. Quando utilizzate il menu Formato, il controllo con le maniglie bianche funge da base per allineare o ridimensionare gli altri controlli selezionati.

Gestire l'input da tastiera

Molti utenti preferiscono navigare nella finestra di dialogo usando la tastiera: Premendo Tab o Maiusc + Tab si passa ciclicamente da un controllo all'altro, mentre premendo un tasto di scelta rapida si attiva istantaneamente un particolare controllo.

Per assicurarvi che la vostra finestra di dialogo funzioni correttamente per gli utenti che amano usare la tastiera, dovete prestare attenzione a due problemi:

- l'ordine di tabulazione;
- i tasti di scelta rapida (Accelerator).

Cambiare l'ordine di tabulazione

L'ordine di tabulazione determina l'ordine in cui i controlli vengono attivati quando l'utente preme Tab o Maiusc + Tab. Determina anche quale controllo ha il *focus* all'inizio, cioè quale controllo è attivo quando appare la finestra di dialogo. Se un utente, per esempio, sta inserendo del testo in una TextBox, la TextBox ha lo stato attivo. Se l'utente fa clic su un OptionButton, tale OptionButton ha lo stato attivo. Il primo controllo nell'ordine di tabulazione è quello che ha lo stato attivo nel momento in cui Excel visualizza per la prima volta una finestra di dialogo.

Per impostare l'ordine di tabulazione, selezionate Visualizza ➪ Ordine di tabulazione. Potete anche fare clic destro sulla finestra di dialogo e scegliere dal menu contestuale il comando Ordine di tabulazione. In entrambi i casi, Excel visualizza la finestra di dialogo Ordine di tabulazione, rappresentata nella Figura 17.18.

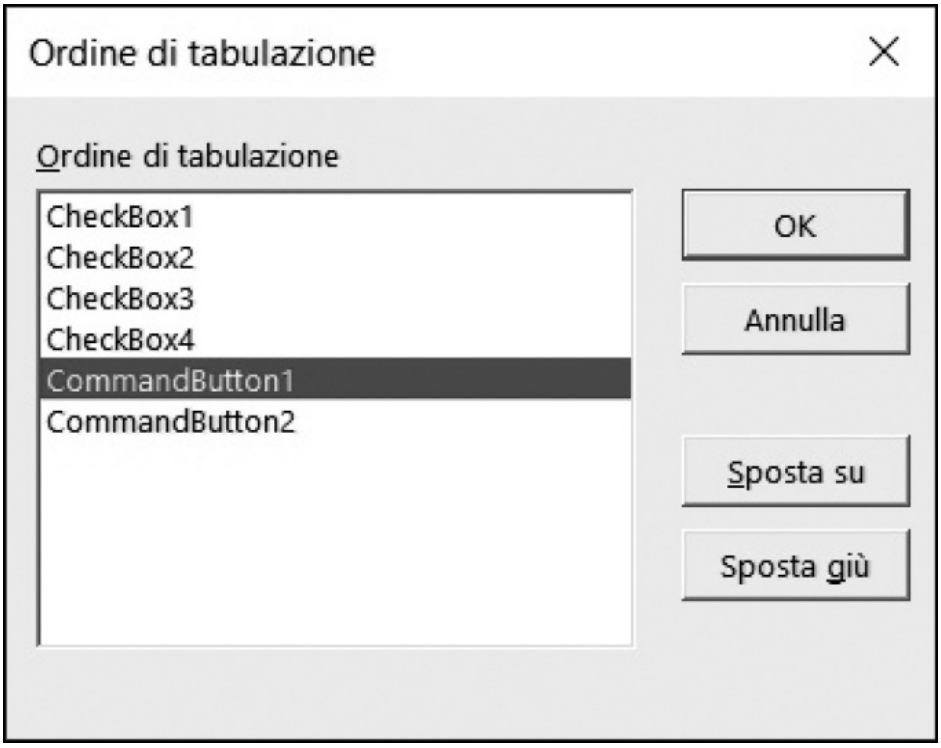

FIGURA 17.18 La finestra di dialogo Ordine di tabulazione.

La finestra di dialogo Ordine di tabulazione elenca tutti i controlli presenti nella UserForm. L'ordine di tabulazione nella UserForm corrisponde all'ordine degli oggetti nell'elenco. Per modificare l'ordine di tabulazione di un controllo, selezionatelo nell'elenco e fate clic sul pulsante Sposta su o Sposta giù. Potete anche scegliere più controlli (fate clic su di essi tenendo premuto Maiusc o Ctrl) e spostarli tutti in una volta.

CONSIGLIO

Anziché utilizzare la finestra di dialogo Ordine di tabulazione, potete impostare la posizione di un controllo nell'ordine di tabulazione utilizzando la finestra Proprietà. Il primo controllo nell'ordine di tabulazione ha la proprietà TabIndex pari a 0. Se desiderate rimuovere un controllo dall'ordine di tabulazione, impostate la proprietà TabStop su False.

RICORDA!

Alcuni controlli (come Frame o MultiPage) fungono da contenitori per altri controlli. I controlli all'interno di un controllo contenitore hanno un proprio ordine di tabulazione. Per impostare l'ordine di tabulazione per un gruppo di OptionButton all'interno di un controllo Frame, selezionate il controllo Frame prima di selezionare il comando Visualizza ➪ Ordine di tabulazione.

Impostare i tasti di scelta rapida

Normalmente, è opportuno assegnare ai controlli della finestra di dialogo un *tasto di scelta rapida*. Per farlo, inserite una lettera per la proprietà Accelerator nella finestra Proprietà. Se un controllo non ha una proprietà Accelerator (per esempio una TextBox), è comunque possibile consentire l'accesso diretto alla tastiera utilizzando un controllo Label. Assegnate un tasto di scelta rapida alla Label e ponetela appena prima della textBox nell'ordine di tabulazione.

La Figura 17.19 mostra una UserForm con tre TextBox. Le etichette che descrivono le TextBox hanno tasti di scelta rapida e ciascuna etichetta precede il relativo TextBox nell'ordine di tabulazione. Premendo Alt + D, per esempio, si attiva la casella di testo accanto all'etichetta Dipartimento.

Provare una UserForm

Il VBE offre tre modi per provare l'aspetto di una UserForm senza richiamarla da una routine VBA.

» Scegliete il comando Esegui ➪ Esegui Sub/UserForm.
» Premete F5.
» Fate clic sul pulsante Esegui Sub/UserForm nella barra degli strumenti Standard.

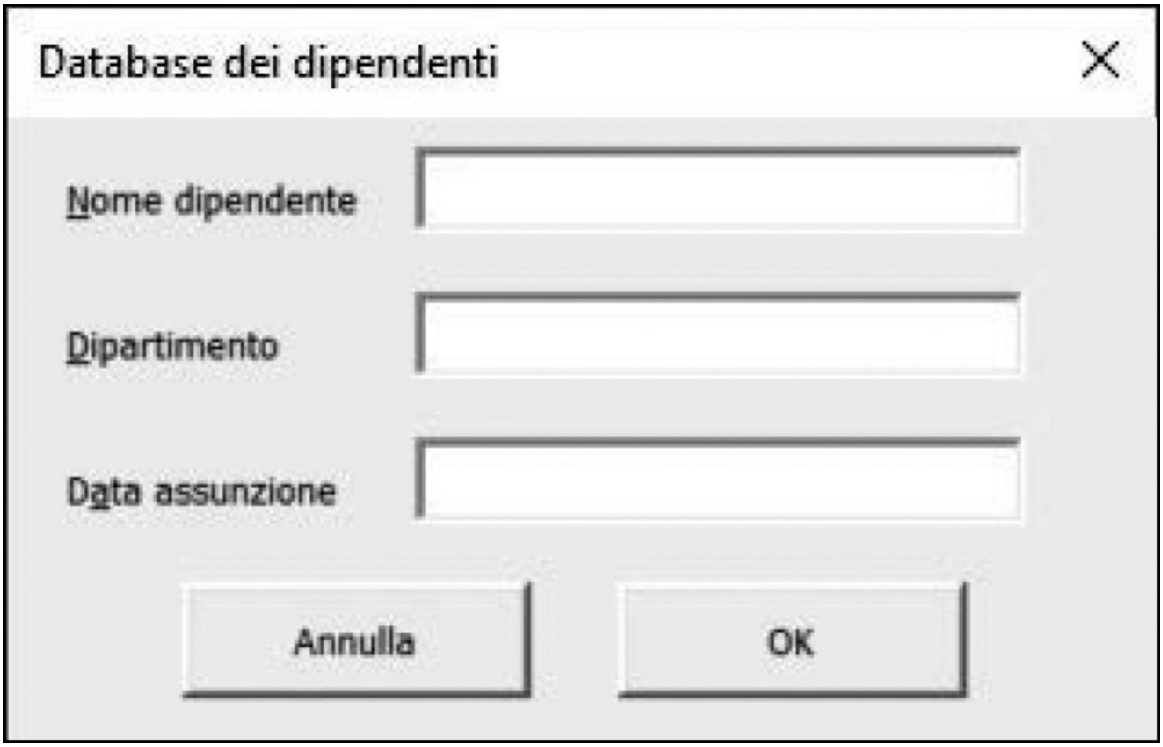

FIGURA 17.19 Potete utilizzare delle Label per fornire accesso diretto ai controlli che non dispongono di tasti di scelta rapida.

Quando viene visualizzata una finestra di dialogo in questa modalità di test, potete provare l'ordine di tabulazione e i tasti di scelta rapida.

L'estetica delle finestre di dialogo

Le finestre di dialogo possono essere belle, brutte o così-così. Una finestra di dialogo di bell'aspetto è facile da interpretare, ha controlli ben dimensionati e allineati e la sua funzione è perfettamente chiara all'utente. Le finestre di dialogo di brutto aspetto confondono l'utente, hanno controlli disallineati e danno l'impressione che lo sviluppatore non l'abbia pianificata a modo.

Cercate di limitare il numero di controlli presenti. Se avete bisogno di usare molti controlli – generalmente più di dieci – considerate l'utilizzo di un controllo MultiPage, per suddividere in più pagine l'attività che l'utente deve eseguire secondo passaggi più logici (e più piccoli).

Una buona regola da seguire è provare a far somigliare le finestre di dialogo alle finestre di dialogo incorporate di Excel. Man mano che acquisirete esperienza con la realizzazione delle finestre di dialogo, potrete replicare quasi tutte le funzionalità delle finestre di dialogo di Excel.

Capitolo 18

Tecniche e trucchi per le UserForm

IN QUESTO CAPITOLO

- **Creare una finestra di dialogo: un esempio pratico**
- **Lavorare con i controlli ListBox**
- **Lasciare che l'utente selezioni un intervallo da una UserForm**
- **Un indicatore di avanzamento per le operazioni di lunga durata**
- **Creare una finestra di dialogo in primo piano**
- **Visualizzare un grafico in una UserForm**
- **Presentare un pratico elenco di controllo per la creazione di finestre di dialogo**

I capitoli precedenti hanno mostrato come inserire una UserForm (che contiene una finestra di dialogo personalizzata), come aggiungere controlli alla UserForm e come modificare alcune delle proprietà dei controlli. Queste possibilità, tuttavia, da sole non vi saranno utili, a meno che non comprendiate come utilizzare la UserForm nel vostro codice VBA. Questo capitolo fornisce questi dettagli mancanti e presenta alcune tecniche e trucchi utili per operare.

Usare le finestre di dialogo

Quando nell'applicazione utilizzate una finestra di dialogo personalizzata, di solito dovete anche scrivere del codice VBA che fa quanto segue.

- Inizializza i controlli della UserForm. Per esempio, potete scrivere il codice che imposta i valori predefiniti dei controlli.
- Visualizza la finestra di dialogo utilizzando il metodo Show dell'oggetto UserForm.
- Risponde agli eventi che si verificano per i vari controlli, per esempio un clic su un pulsante di comando.
- Convalida le informazioni fornite dall'utente (sempre che l'utente non abbia fatto clic su Annulla nella finestra di dialogo). Questo passaggio non è sempre necessario.
- Svolge una specifica azione con le informazioni fornite dall'utente (se l'informazione è valida).
- Chiude la UserForm utilizzando il metodo Unload.

Un esempio di UserForm

Questo esempio mostra i punti descritti nel paragrafo precedente. In questo esempio, utilizzate una finestra di dialogo per ottenere due informazioni: nome e sesso di una persona. La finestra di dialogo utilizza un controllo TextBox per ottenere il nome e tre OptionButton per ottenere il sesso (Maschio, Femmina o Sconosciuto). Le informazioni raccolte nella finestra di dialogo vengono quindi inviate alla successiva riga vuota in un foglio di lavoro.

Creare la finestra di dialogo

La Figura 18.1 mostra l'aspetto della UserForm completata, al termine di questo esempio. Per migliori risultati, iniziate con una nuova cartella di lavoro contenente un solo un foglio di lavoro. Quindi seguite questi passaggi.

1. **Premete Alt + F11 per attivare il VBE.**
2. **Nel riquadro Progetto, selezionate la cartella di lavoro vuota e selezionate il comando Inserisci ➪ UserForm.**

   Al progetto verrà aggiunta una UserForm vuota.

3. **Modificate la proprietà Caption della UserForm per richiedere il nome e il sesso.**
   Se la finestra Proprietà non è visibile, premete F4.

FIGURA 18.1 Questa finestra di dialogo chiede all'utente di specificare il nome e il sesso.

Questa finestra di dialogo ha otto controlli, con le proprietà impostate nel modo seguente.

» Etichetta (Label)

Proprietà	Valore
Accelerator	N
Caption	Nome
TabIndex	0

» Casella di testo (TextBox)

Proprietà	Valore
Name	TextName
TabIndex	1

» Cornice (Frame)

Proprietà	Valore
Caption	Sesso
TabIndex	2

» Pulsante di opzione (OptionButton)

Proprietà	Valore
Accelerator	M
Caption	Maschio
Name	OptionMale
TabIndex	0

» Un altro Pulsante di opzione (OptionButton)

Proprietà	Valore
Accelerator	F
Caption	Femmina
Name	OptionFemale
TabIndex	1

» Un altro Pulsante di opzione (OptionButton)

Proprietà	Valore
Accelerator	U

Caption	Sconosciuto
Name	OptionUnknown
TabIndex	2
Value	True

» Pulsante di comando (CommandButton)

Proprietà	Valore
Caption	Enter
Default	True
Name	EnterButton
TabIndex	b

» Pulsante di comando (CommandButton)

Proprietà	Valore
Caption	Close
Cancel	True
Name	CloseButton
TabIndex	4

Se state seguendo queste istruzioni sul vostro computer (e dovreste farlo), dedicate alcuni minuti a creare questa UserForm utilizzando le informazioni precedenti. Assicuratevi di creare l'oggetto Frame prima di aggiungere gli OptionButton.

In alcuni casi, è più facile copiare un controllo esistente piuttosto che crearne uno nuovo. Per copiare un controllo, premete Ctrl mentre lo trascinate.

CONSIGLIO

Se preferite la strada più semplice e veloce, potete scaricare l'esempio (in inglese) dal sito web di questo libro.

Scrivere il codice per visualizzare la finestra di dialogo

Dopo aver aggiunto i controlli alla UserForm, il passo successivo è sviluppare un codice VBA per visualizzare questa finestra di dialogo.

1. **Nella finestra di VBE, selezionate il comando Inserisci ➪ Modulo per inserire un modulo VBA.**
2. **Inserite la seguente macro:**

```
Sub GetData()
     UserForm1.Show
End Sub
```

Questa breve routine utilizza il metodo Show dell'oggetto UserForm per visualizzare la finestra di dialogo.

Rendere disponibile la macro

I passaggi successivi offrono all'utente un modo semplice per eseguire la routine.

1. **Attivate Excel.**
2. **Selezionate il comando Sviluppo ➪ Controlli ➪ Inserisci e fate clic sull'icona Pulsante nella sezione Controlli modulo.**
3. **Trascinatela nel foglio di lavoro per creare il pulsante.**

   Verrà visualizzata la finestra di dialogo Assegna macro.
4. **Assegnate al pulsante la macro GetData.**
5. **Modificate la didascalia del pulsante in Inserimento dati.**

CONSIGLIO

Se volete essere davvero fantasiosi, potete aggiungere un'icona alla barra di accesso rapido. Quindi facendo clic sull'icona potrete eseguire la macro GetData. Per configurarla, fate clic destro sulla barra di accesso rapido e selezionate il comando Personalizza barra di accesso rapido, che apre la scheda Barra di accesso rapido della finestra di dialogo Opzioni di Excel. Dal menu a discesa Scegli comandi da, selezionate Macro. Quindi selezionate la macro GetData e fate clic su Aggiungi. Se lo desiderate, potete fare clic sul pulsante Modifica, e modificare l'icona. Inoltre potreste rendere visibile l'icona della barra di accesso rapido solo quando viene attivata la cartella di lavoro appropriata. Prima di aggiungere la macro, utilizzate il controllo a discesa nella parte superiore destra della finestra di dialogo Opzioni di Excel, per specificare il nome della cartella di lavoro, anziché lasciare Per tutti i documenti (impostazione predefinita).

Provare la finestra di dialogo

Per provare la finestra di dialogo impiegate la seguente procedura.

1. **Fate clic sul pulsante Inserimento dati, nel foglio di lavoro o fate clic sull'icona nella barra di accesso rapido se l'avete creata.**

   Apparirà la finestra di dialogo, rappresentata nella Figura 18.2.

2. **Inserite del testo nella casella di modifica.**
3. **Fate clic su Invio o Chiudi.**

   Non succede nulla, il che è comprensibile perché non avete ancora creato alcuna routine.

4. **Fate clic sul pulsante X sulla barra del titolo della finestra di dialogo per chiudere la finestra di dialogo.**

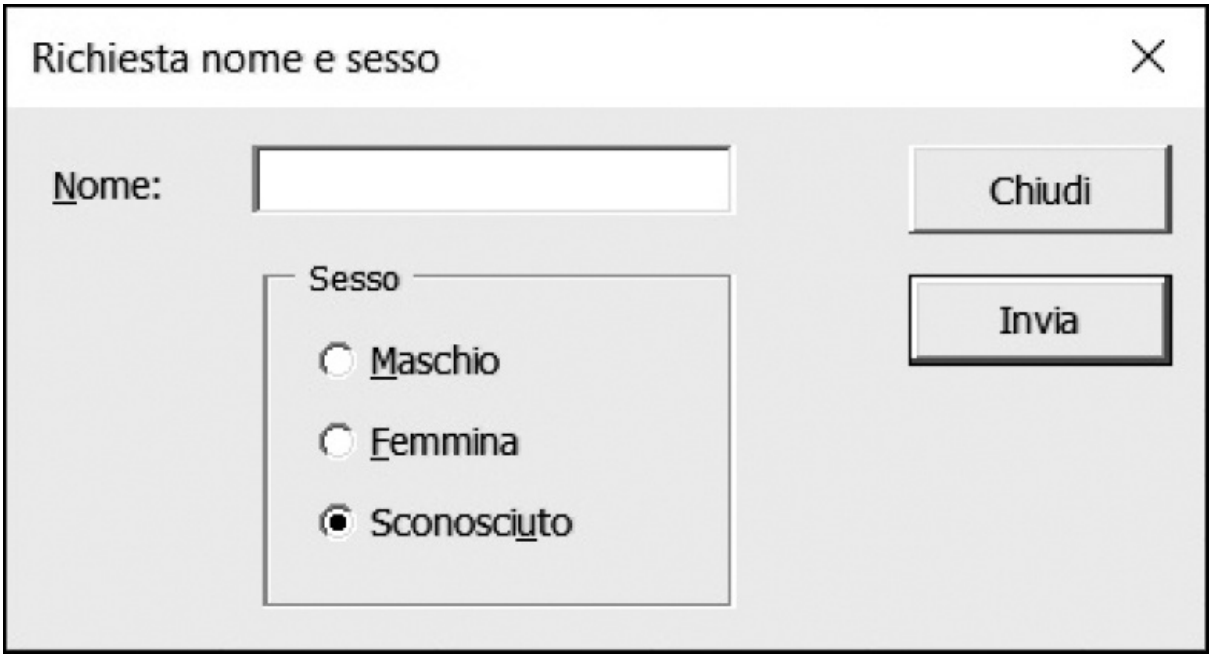

FIGURA 18.2 L'esecuzione della routine GetData visualizza la finestra di dialogo.

Aggiungere le routine di gestione degli eventi

Spesso, quando si verificano determinati eventi delle finestre di dialogo, intendete attivare determinate routine. Per esempio, potreste voler eseguire una routine quando aprite una finestra di dialogo. Oppure volete che Excel salvi la cartella di lavoro ogni volta che si chiude la finestra di dialogo.

Questo paragrafo vi guida lungo un esempio di routine che gestisce gli eventi della finestra di dialogo.

Iniziate seguendo questi passaggi.

1. **Premete Alt + F11 per richiamare il VBE e assicuratevi che sia visualizzata la UserForm.**
2. **Fate doppio clic sul pulsante Chiudi della UserForm.**

   Il VBE attiva la finestra Codice per la UserForm e fornisce una routine vuota denominata CloseButton_Click.

3. **Modificate la routine come segue:**

```
Private Sub CloseButton_Click()
     Unload UserForm1
End Sub
```

Questa routine, che viene eseguita quando l'utente fa clic sul pulsante Chiudi, elimina semplicemente la finestra di dialogo dalla memoria.

4. **Premete Maiusc + F7 per visualizzare nuovamente la UserForm1.**
5. **Fate doppio clic sul pulsante Invio e immettete la seguente routine:**

```
Private Sub EnterButton_Click()
     Dim NextRow As Long

'   Si assicura che sia attivo il Foglio1
     Sheets("Foglio1").Activate

'   Determina la successiva riga vuota
     NextRow = Application.WorksheetFunction. _
         CountA(Range("A:A")) + 1

     '   Trasferisce il nome
     Cells(NextRow, 1) = TextName.Text

'   Trasferisce il sesso
     If OptionMale Then Cells(NextRow, 2) = "Maschio"
     If OptionFemale Then Cells(NextRow, 2) = "Femmina"
     If OptionUnknown Then Cells(NextRow, 2) = "Sconosciuto"

'   Cancella i controlli per il successivo inserimento
     TextName.Text = ""
     OptionUnknown = True
     TextName.SetFocus
End Sub
```

6. **Attivate Excel ed eseguite nuovamente la routine facendo clic sul pulsante Inserimento dati.**

   La finestra di dialogo ora funzionerà. La Figura 18.3 mostra il suo aspetto in azione.

Ecco come funziona la routine EnterButton_Click.

1. Il codice si assicura che sia attivo il foglio di lavoro corretto (Foglio1).
2. Utilizza quindi la funzione COUNTA di Excel per contare il numero di voci nella colonna A e per determinare la cella vuota successiva nella colonna.

FIGURA 18.3 Uso della finestra di dialogo personalizzata per l'immissione dei dati.

3. La routine trasferisce il testo dalla TextBox alla colonna A.
4. Utilizza una serie di istruzioni If per determinare quale OptionButton è stato selezionato e scrive nella colonna B il testo appropriato (Maschio, Femmina o Sconosciuto).
5. La finestra di dialogo viene ripristinata per prepararla per la voce successiva.

Notate che facendo clic sul pulsante Invio non si chiude la finestra di dialogo, poiché l'utente probabilmente vorrà immettere altri dati. Per terminare l'immissione dei dati, fate clic sul pulsante Chiudi.

Convalidare i dati

Provando e riprovando questa routine, scoprirete che la macro ha un piccolo problema: non garantisce che l'utente inserisca effettivamente un nome nella Text Box. Il seguente codice, da inserire nella routine EnterButton_Click prima che il testo venga trasferito al foglio di lavoro, si assicura che l'utente inserisca del testo nella TextBox. Se la casella di testo è vuota, visualizza un messaggio e la routine si interrompe. Tuttavia, la finestra di dialogo rimane aperta e l'utente può correggere il problema.

```
'   Si assicura che sia stato inserito un nome
     If TextName.Text = "" Then
         MsgBox "Introdurre un nome."
         Exit Sub
     End If
```

Ora la finestra di dialogo funziona

Dopo aver apportato queste modifiche, la finestra di dialogo funziona perfettamente. Nella vita reale, probabilmente dovreste raccogliere maggiori informazioni, tuttavia, si applicheranno gli stessi principi di base. Dovrete solo gestire più controlli UserForm.

Un'altra cosa da ricordare: se i dati non iniziano nella riga 1 o se l'area dati contiene righe vuote, il conteggio per la variabile NextRow sarà errato. La funzione COUNTA conta il numero di celle nella colonna A e l'ipotesi è che non ci siano celle vuote sopra il cognome, nella colonna. Ecco un altro modo per determinare la riga vuota successiva:

```
NextRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
```

L'istruzione simula l'attivazione dell'ultima cella nella colonna A, la pressione di Fine, la pressione del tasto freccia su e poi lo spostamento di una riga verso il basso. Se provate a farlo manualmente, il cursore della cella si troverà nella successiva cella vuota della colonna A, anche se l'area dati non inizia nella riga 1 e contiene righe vuote.

Un esempio di ListBox

Le ListBox sono controlli utili, ma il loro uso può essere un po' complicato. Prima di visualizzare una finestra di dialogo che utilizza una ListBox, è necessario riempire la ListBox con degli elementi. Quindi, quando la finestra di dialogo viene chiusa, è necessario determinare quale elemento (o quali elementi) ha selezionato l'utente.

CONSIGLIO

Quando avete a che fare con le ListBox, è necessario conoscere le seguenti proprietà e i seguenti metodi.

- **AddItem:** questo metodo si utilizza per aggiungere un elemento a una ListBox.
- **ListCount:** questa proprietà restituisce il numero di elementi presenti nella ListBox.
- **ListIndex:** questa proprietà restituisce il numero di indice dell'elemento selezionato o imposta l'elemento selezionato (solo selezioni singole). Il primo elemento ha un ListIndex pari a 0 (non 1).
- **MultiSelect:** questa proprietà determina se l'utente può selezionare più di un elemento dalla ListBox.
- **RemoveAllItems:** utilizzate questo metodo per rimuovere tutti gli elementi da un controllo ListBox.
- **Selected:** questa proprietà restituisce una matrice che indica gli elementi selezionati (applicabile solo quando sono consentite più selezioni).
- **Value:** questa proprietà restituisce l'elemento selezionato in una ListBox.

CONSIGLIO

La maggior parte dei metodi e delle proprietà che funzionano con le ListBox funzionano anche con le ComboBox. Quindi, dopo aver capito come gestire le ListBox, potete trasferire queste conoscenze al vostro lavoro con le ComboBox.

Compilare il contenuto di una ListBox

Per semplificare le cose, iniziate con una cartella di lavoro vuota. L'esempio in questo paragrafo presuppone quanto segue.

- Avete aggiunto una UserForm.
- La UserForm contiene un controllo ListBox denominato ListBox1.
- La UserForm ha un controllo CommandButton denominato OKButton.
- La UserForm ha un controllo CommandButton denominato CancelButton, che ha la seguente routine di gestione degli eventi:

```
Private Sub CancelButton_Click()
     Unload UserForm1
End Sub
```

Questa routine è archiviata nella routine di inizializzazione per la UserForm. Seguite questa procedura.

1. **Selezionate la UserForm e premete F7 per attivare la sua finestra Codice.**

   Il VBE visualizza la finestra Codice per il modulo ed è pronto per l'inserimento del codice per l'evento Initialize.

2. **Dall'elenco a discesa Routine nella parte superiore della finestra Codice, selezionate Initialize.**

3. **Aggiungete il codice di inizializzazione per il modulo:**

```
Sub UserForm_Initialize()
'   Compila la casella di riepilogo
      With ListBox1
            .AddItem "Gennaio"
            .AddItem "Febbraio"
            .AddItem "Marzo"
            .AddItem "Aprile"
            .AddItem "Maggio"
            .AddItem "Giugno"
            .AddItem "Luglio"
            .AddItem "Agosto"
            .AddItem "Settembre"
            .AddItem "Ottobre"
            .AddItem "Novembre"
            .AddItem "Dicembre"
      End With

     '   Seleziona il primo elemento dell'elenco
     ListBox1.ListIndex = 0
End Sub
```

   Questa routine di inizializzazione viene eseguita automaticamente ogni volta che viene caricata la UserForm. Pertanto, quando utilizzate il metodo Show della UserForm, il codice popola la ListBox con 12 elementi, ciascuno aggiunto tramite il metodo AddItem.

4. **Inserite un modulo VBA e digitate questa breve routine Sub per visualizzare la finestra di dialogo:**

```
Sub ShowList()
     UserForm1.Show
End Sub
```

Determinare l'elemento selezionato

Il codice precedente mostra semplicemente una finestra di dialogo con una ListBox piena di nomi di mesi. Ciò che manca è una routine per determinare quale elemento è selezionato nella ListBox.

Fate doppio clic su OKButton e aggiungete la seguente routine OKButton_Click:

```
Private Sub OKButton_Click()
     Dim Msg As String
     Msg = "Avete selezionato l'elemento numero "
     Msg = Msg & ListBox1.ListIndex
     Msg = Msg & vbNewLine
     Msg = Msg & ListBox1.Value
     MsgBox Msg
     Unload UserForm1
End Sub
```

Questa routine apre una finestra MsgBox con il numero dell'elemento e l'elemento selezionato.

Se non è stato selezionato alcun elemento nella ListBox, la proprietà ListIndex restituirà -1. Tuttavia, questo non sarà mai il caso di questa particolare ListBox, poiché il codice presente nella routine UserForm_Initialize ha selezionato il primo elemento. È impossibile deselezionare un elemento senza selezionarne un altro. Quindi ci sarà *sempre* un

elemento selezionato, se l'utente non seleziona effettivamente un mese.

La Figura 18.4 mostra il suo aspetto.

FIGURA 18.4 Determinare quale oggetto è selezionato in una ListBox.

PER I PIÙ CURIOSI

Il primo elemento di una ListBox ha ListIndex pari a 0, non 1 (come potreste aspettarvi). È sempre così, anche se utilizzate un'istruzione Option Base 1 per modificare il limite inferiore predefinito per le matrici.

Questo esempio è disponibile, in originale inglese, sul sito web di questo libro.

Determinare le selezioni multiple

Se la ListBox è impostata in modo che l'utente possa selezionare più di un elemento, scoprirete che la proprietà ListIndex restituisce solo l'ultimo elemento selezionato. Per determinare tutti gli elementi selezionati, è necessario utilizzare la proprietà Selected, che contiene una matrice.

RICORDA!

Per consentire selezioni multiple in un controllo ListBox, impostate la proprietà MultiSelect su 1 o 2. Potete farlo in fase di progettazione utilizzando la finestra Proprietà o in fase di esecuzione utilizzando un'istruzione VBA come questa:

```
UserForm1.ListBox1.MultiSelect = 1
```

La proprietà MultiSelect ha tre possibili impostazioni. Il significato di ciascuna è mostrato nella Tabella 18.1.

TABELLA 18.1 **Impostazioni per la proprietà MultiSelect**

Valore	Costante VBA	Significato
0	fmMultiSelectSingle	Potete selezionare solo un solo elemento.
1	fmMultiSelectMulti	Facendo clic su un elemento o premendo la barra spaziatrice selezionate o deselezionate un elemento nell'elenco.
2	fmMultiSelectExtended	Gli oggetti vengono aggiunti o rimossi dall'insieme di selezione tenendo premuto il tasto Maiusc o Ctrl mentre fate clic sugli elementi.

La seguente routine apre una finestra MsgBox che elenca tutti gli elementi selezionati di una ListBox. La Figura 18.5 mostra un esempio.

```
Private Sub OKButton_Click()
     Dim Msg As String
     Dim i As Long
     Dim Counter As Long
     Msg = "Avete selezionato" & vbNewLine
     For i = 0 To ListBox1.ListCount - 1
         If ListBox1.Selected(i) Then
              Counter = Counter + 1
              Msg = Msg & ListBox1.List(i) & vbNewLine
         End If
     Next i
     If Counter = 0 Then Msg = Msg & "(nulla)"
     MsgBox Msg
     Unload UserForm1
End Sub
```

Questa routine utilizza un ciclo For-Next per scorrere ogni elemento nella ListBox. Notate che il ciclo inizia con l'elemento 0 (il primo elemento) e termina con l'ultimo elemento (determinato dal valore della proprietà ListCount meno 1). Se la proprietà Selected di un elemento è True, significa che l'elemento dell'elenco è stato selezionato. Il codice utilizza anche una variabile (contatore) per tenere traccia di quanti elementi sono stati selezionati. Un'istruzione If-Then cambia messaggio se non è stato selezionato nulla.

FIGURA 18.5 Determinare gli elementi selezionati in una ListBox che consente selezioni multiple.

CONSIGLIO

Questo esempio è disponibile, in originale inglese, sul sito web di questo libro.

Selezionare un intervallo

In alcuni casi, potreste volere che l'utente selezioni un intervallo mentre è visualizzata una finestra di dialogo. Un esempio di questo tipo si verifica nella finestra di dialogo Crea tabella, che viene visualizzata quando selezionate Home ➪ Inserisci ➪ Tabelle ➪ Tabella. La finestra di dialogo Crea tabella ha un controllo dedicato alla selezione di un intervallo che contiene la proposta di Excel relativa all'intervallo da convertire, ma è possibile utilizzare questo controllo anche per modificare l'intervallo, selezionando le celle sul foglio di lavoro.

Per consentire la selezione di un intervallo nella finestra di dialogo, basta aggiungere un controllo RefEdit. Come si vede nella Figura 18.6, l'esempio seguente presenta una finestra di dialogo, dove l'intervallo attualmente selezionato è presentato all'interno di un controllo RefEdit. Si tratta del blocco di celle non vuote che contiene la cella attiva. L'utente può accettare questo intervallo o modificarlo. Al termine, quando l'utente farà clic su OK, la routine applicherà il grassetto all'intervallo.

Questo esempio presume quanto segue.

- Avete una UserForm denominata UserForm1.
- La UserForm contiene un controllo CommandButton denominato OKButton.

» La UserForm contiene un controllo CommandButton denominato CancelButton.
» La UserForm contiene un controllo RefEdit denominato RefEdit1.

FIGURA 18.6 Questa finestra di dialogo consente all'utente di selezionare un intervallo.

Il codice, mostrato di seguito, è memorizzato in un modulo VBA. Questo codice esegue due operazioni: inizializza la finestra di dialogo assegnando l'indirizzo della regione corrente al controllo RefEdit e visualizza la UserForm.

```
Sub BoldCells()
'   Esce se non è attivo un foglio di lavoro
     If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub

'   Seleziona la regione corrente
     ActiveCell.CurrentRegion.Select

'   Inizializza il controllo RefEdit
     UserForm1.RefEdit1.Text = Selection.Address

'   Mostra la finestra di dialogo
     UserForm1.Show
End Sub
```

Quando l'utente fa clic sul pulsante OK viene eseguita la seguente routine, la quale esegue un semplice controllo degli errori per assicurarsi che l'intervallo specificato nel controllo RefEdit sia valido.

```
Private Sub OKButton_Click()
     On Error GoTo BadRange
     Range(RefEdit1.Text).Font.Bold = True
     Unload UserForm1
     Exit Sub
BadRange:
     MsgBox "L'intervallo specificato non è valido."
End Sub
```

Se si verifica un errore (in generale una specifica di intervallo non valida nel controllo RefEdit), il codice salta all'etichetta BadRange e viene visualizzata una finestra MsgBox. La finestra di dialogo rimane però aperta, in modo che l'utente possa selezionare un altro intervallo.

CONSIGLIO

Se, l'*unica* funzione eseguita dalla vostra UserForm, è chiedere all'utente di selezionare un intervallo, potete semplificare le cose usando il metodo Application.InputBox (vedi il Capitolo 15).

Utilizzare più insiemi di OptionButton

La Figura 18.7 mostra una finestra di dialogo personalizzata con tre insiemi di OptionButton. Se la vostra UserForm contiene più di un insieme di OptionButton, assicuratevi che ogni insieme funzioni come un gruppo. Potete farlo in due modi.

- Racchiudere ciascun insieme di OptionButton in un controllo Frame. Questo approccio è il più semplice e migliora la chiarezza della finestra di dialogo. È più semplice aggiungere la cornice prima di aggiungere gli OptionButton. Tuttavia, potete anche trascinare degli OptionButton esistenti in un Frame.
- Assicurarsi che ogni insieme di OptionButton abbia la stessa proprietà GroupName (si specifica nella finestra Proprietà). Se gli OptionButton sono contenuti in un Frame, non dovrete preoccuparvi della proprietà GroupName.

RICORDA!

Solo un OptionButton in un gruppo può avere il valore True. Per specificare un'opzione predefinita per un insieme di OptionButton, basta impostare a True la proprietà Value per l'elemento predefinito. Potete farlo direttamente nella finestra Proprietà o usando del codice VBA:

```
UserForm1.OptionButton1.Value = True
```

CONSIGLIO

Questo esempio è disponibile, in originale inglese, sul sito web di questo libro. Include anche il codice che visualizza le opzioni selezionate quando l'utente fa clic su OK.

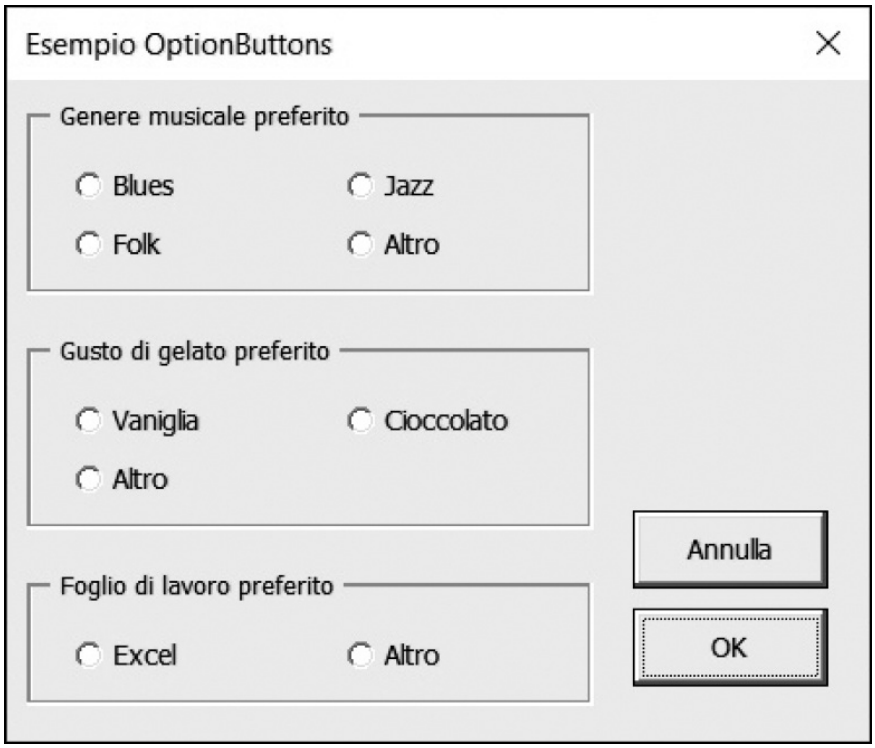

FIGURA 18.7 Questa finestra di dialogo contiene tre insiemi di controlli OptionButton.

Utilizzare uno SpinButton e una TextBox

Un controllo SpinButton consente all'utente di specificare un numero facendo clic su delle frecce. Questo controllo è composto solo da frecce (senza testo), quindi di solito c'è bisogno di un metodo per visualizzare il numero selezionato. Una possibilità consiste nell'usare un controllo Label, che però presenta uno svantaggio: l'utente non può digitare del testo in una Label. Una scelta migliore consiste nell'usare una TextBox.

I controlli SpinButton e TextBox formano una coppia naturale, ed Excel li utilizza frequentemente. Per esempio, nella finestra di dialogo Imposta pagina di Excel trovate alcuni esempi. Idealmente, lo SpinButton e la sua TextBox sono sempre sincronizzati: se l'utente fa clic sullo SpinButton, nella TextBox dovrebbe apparire il valore individuato dallo SpinButton. E se l'utente inserisce un valore direttamente nella TextBox, lo SpinButton dovrebbe assumere quel valore. La Figura 18.8 mostra una finestra di dialogo personalizzata con uno SpinButton e una TextBox.

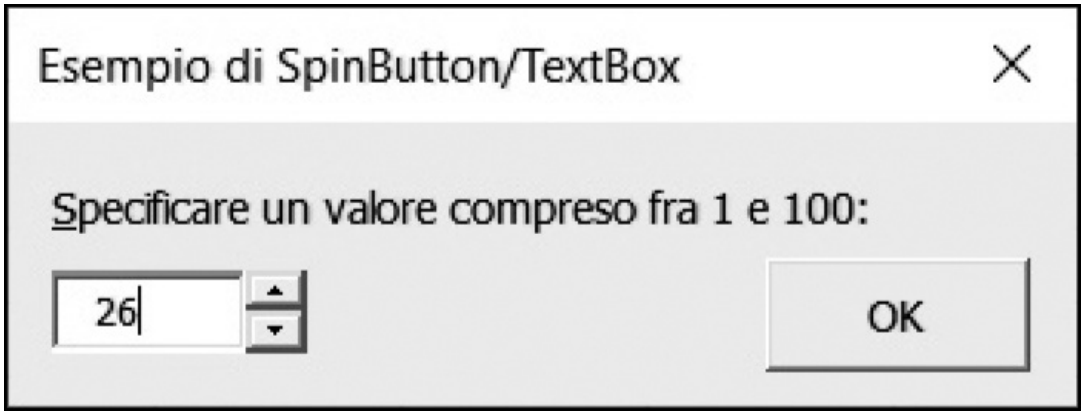

FIGURA 18.8 Una UserForm con uno SpinButton e la suaTextBox.

Questa UserForm contiene i seguenti controlli.

- Un oggetto SpinButton denominato SpinButton1, con la proprietà Min impostata a 1 e la proprietà Max impostata a 100.
- Una TextBox denominata TextBox1, posizionata a sinistra dello SpinButton.
- Un CommandButton denominato OKButton.

Quella che segue è la routine di gestione degli eventi per lo SpinButton. Questa routine gestisce l'evento Change, che si attiva ogni volta che viene modificato il valore di SpinButton. Quando il valore dello SpinButton cambia (quando viene selezionato), questa routine assegna tale valore alla TextBox. Per creare questa routine, fate doppio clic sullo SpinButton per attivare la finestra Codice per la UserForm. Quindi inserite il seguente codice:

```
Private Sub SpinButton1_Change()
     TextBox1.Text = SpinButton1.Value
End Sub
```

Il gestore di eventi per la TextBox, elencato in seguito, è un po' più complicato. Per creare questa routine, fate doppio clic sulla TextBox per attivare la finestra Codice per la UserForm. Questa routine viene eseguita ogni volta che l'utente modifica il testo nella TextBox.

```
Private Sub TextBox1_Change()
     Dim NewVal As Long

     NewVal = Val(TextBox1.Text)
     If NewVal >= SpinButton1.Min And _
         NewVal <= SpinButton1.Max Then _
         SpinButton1.Value = NewVal
End Sub
```

Questa routine utilizza una variabile che conserva il testo specificato nella TextBox (convertito in un valore con la funzione Val). Quindi lo controlla per assicurarsi che il valore sia compreso nell'intervallo corretto. In tal caso, lo SpinButton viene impostato sul valore indicato nella TextBox. L'effetto netto è che il valore dello SpinButton è sempre uguale al valore indicato nella TextBox (sempre che il valore dello SpinButton sia nell'intervallo corretto).

PER I PIÙ CURIOSI

Se premete una sola volta F8 nel codice in modalità di debug, noterete che quando viene eseguita la riga SpinButton1.Value = NewVal, viene immediatamente attivato l'evento Change dello SpinButton. A sua volta, l'evento SpinButton1_Change imposta il valore della TextBox1. Fortunatamente, questo a sua volta non attiva l'evento TextBox1_Change, perché il suo valore non è stato modificato dall'evento SpinButton1_Change. Ma potete immaginare che questo effetto possa causare risultati sorprendenti nella vostra UserForm.

Confusi? Ricordate che se il vostro codice cambia il valore di un controllo, si attiverà l'evento Change di quel controllo.

CONSIGLIO

Questo esempio è disponibile, in originale inglese, sul sito web del libro. Vi troverete anche alcuni altri elementi e accorgimenti utili.

Utilizzare una UserForm come indicatore di avanzamento

Se avete una macro che richiede molto tempo di esecuzione, potete visualizzare un indicatore di avanzamento, in modo che gli utenti non pensino che Excel sia bloccato. Per creare un indicatore di avanzamento gradevole, come quello rappresentato nella Figura 18.9, potete utilizzare una UserForm. Tuttavia, tale uso delle finestre di dialogo richiede alcuni trucchi.

FIGURA 18.9 Questa UserForm funge da indicatore di avanzamento di una lunga macro.

Creare la finestra di dialogo dell'indicatore di avanzamento

Il primo passo consiste nel creare la UserForm. In questo esempio, la finestra di dialogo visualizza l'avanzamento, mentre una macro inserisce dei numeri casuali nelle celle di 100 colonne e 1.000 righe del foglio di lavoro attivo. Per creare la finestra di dialogo, utilizzate la seguente procedura.

1. **Attivate il VBE e inserite una nuova UserForm.**
2. **Modificate l'etichetta della UserForm in Avanzamento.**
3. **Aggiungete un oggetto Cornice (Frame) e impostate le seguenti proprietà:**

Proprietà	Valore
Caption	0%
Name	FrameProgress
SpecialEffect	2 — fmSpecialEffectSunken
Width	204
Height	28

4. **Aggiungete all'interno del Frame un oggetto Etichetta (Label) e impostate le seguenti proprietà:**

Proprietà	Valore
Name	LabelProgress

BackColor	&H0000C000& (green)
Caption	(nulla)
SpecialEffect	1 — fmSpecialEffectRaised
Width	20
Height	13
Top	5
Left	2

5. **Aggiungete un'altra Etichetta (Label) sopra il Frame e modificate la sua proprietà Caption in Inserimento di numeri casuali...**

   La UserForm dovrebbe avere un aspetto simile a quello rappresentato nella Figura 18.10.

Le routine

Questo esempio utilizza due routine e una variabile a livello di modulo.

» **La variabile a livello di modulo.** Situata in un modulo VBA, questa variabile contiene la copia della UserForm:

```
Dim ProgressIndicator as UserForm1
```

FIGURA 18.10 L' indicatore di progressione di UserForm.

» **EnterRandomNumbers.** Questa routine svolge tutto il lavoro e viene eseguita quando viene visualizzata la UserForm. Notate che richiama la routine UpdateProgress, la quale aggiorna l'indicatore di avanzamento nella finestra di dialogo:

```
Sub EnterRandomNumbers()
'   Inserisce numeri casuali nel foglio di lavoro attivo
     Dim Counter As Long
     Dim RowMax As Long, ColMax As Long
     Dim r As Long, c As Long
     Dim PctDone As Single

'   Crea una copia della UserForm in una variabile
     Set ProgressIndicator = New UserForm1

'   Mostra ProgressIndicator in uno stato non modale
     ProgressIndicator.Show vbModeless
     If TypeName(ActiveSheet) <> "Worksheet" Then
         Unload ProgressIndicator
         Exit Sub
     End If

'   Inserisce i numeri casuali
     Cells.Clear
```

```
      Counter = 1
          RowMax = 200
          ColMax = 50
          For r = 1 To RowMax
               For c = 1 To ColMax
                 Cells(r, c) = Int(Rnd * 1000)
                 Counter = Counter + 1
               Next c
               PctDone = Counter / (RowMax * ColMax)
               Call UpdateProgress(PctDone)
          Next r
          Unload ProgressIndicator
          Set ProgressIndicator = Nothing
End Sub
```

» **UpdateProgress.** Questa routine accetta un argomento e aggiorna nella finestra di dialogo l'indicatore di avanzamento:

```
Sub UpdateProgress(pct)
     With ProgressIndicator
          .FrameProgress.Caption = Format(pct, "0%")
          .LabelProgress.Width = pct * (.FrameProgress _
               .Width - 10)
     End With
'   DoEvents esegue l'aggiornamento della UserForm
     DoEvents
End Sub
```

Come funziona questo esempio

Quando viene eseguita, la routine EnterRandomNumbers carica una copia della UserForm1 nella variabile ProgressIndicator. Quindi imposta a 0 la larghezza dell'etichetta LabelProgress e visualizza la UserForm in uno stato non modale (quindi il codice proseguirà l'esecuzione).

La routine EnterRandomNumber controlla che cos'è il foglio attivo. Se non è un foglio di lavoro, la UserForm (ProgressIndicator) viene chiusa e la routine termina senza alcuna azione. Se il foglio attivo è un foglio di lavoro, la routine procede come segue.

1. Cancella tutte le celle del foglio di lavoro attivo.
2. Controlla tutte le righe e le colonne (specificate dalle variabili RowMax e ColMax) e in ogni cella inserisce un numero casuale.
3. Incrementa la variabile Counter e calcola la percentuale di completamento (che è memorizzata nella variabile PctDone).
4. Richiama la routine UpdateProgress, che mostra la percentuale di completamento modificando la larghezza dell'etichetta LabelProgress e aggiorna la proprietà Caption del controllo Frame.
5. Scarica la UserForm.

L'utilizzo di un indicatore di avanzamento, ovviamente, rallenterà un po' la macro, perché il codice che aggiorna la UserForm aumenta il carico di lavoro. Quindi, se la velocità è un fattore critico, pensateci bene prima di caricare il sistema con un indicatore di avanzamento.

RICORDA!

Se dovete adattare questa tecnica a un uso personale, dovrete capire come determinare i progressi della macro, che variano a seconda del lavoro da essa svolto. Poi richiamate periodicamente la routine UpdateProgress mentre la macro è in esecuzione.

CONSIGLIO

Questo esempio è disponibile, in originale inglese, sul sito web di questo libro.

Creare una finestra di dialogo non modale a schede

Le finestre di dialogo a schede sono utili perché consentono di presentare le informazioni in piccoli blocchi ben organizzati. La finestra di dialogo Formato celle di Excel (che viene visualizzata quando fate clic destro su una cella e selezionate Formato celle) ne è un buon esempio. La finestra di dialogo di questo esempio utilizza tre schede, per aiutare a organizzare alcune delle opzioni di visualizzazione di Excel.

Creare finestre di dialogo a schede è relativamente facile, grazie al controllo MultiPage. La Figura 18.11 mostra una finestra di dialogo personalizzata che utilizza un controllo MultiPage con tre *pagine* o *schede*. Quando l'utente fa clic su una scheda, si attiva una nuova pagina e vengono visualizzati solo i controlli in essa definiti.

La finestra di dialogo è *non modale*. Questo significa che l'utente la può mantenere visualizzata mentre va avanti a lavorare. Ogni controllo ha un effetto immediato, quindi necessita di un pulsante OK. Ecco la routine che visualizza la UserForm, mantenendola in primo piano:

```
Sub ShowDialog()
     UserForm1.Show vbModeless
End Sub
```

FIGURA 18.11 Le tre schede di un controllo MultiPage.

RICORDA!

Quando utilizzate il controllo MultiPage per creare una finestra di dialogo a schede, tenete presente i seguenti punti.

- Utilizzate un solo controllo MultiPage per finestra di dialogo.
- Per rendere sempre visibili alcuni controlli (come OK, Annulla e Chiudi), posizionateli all'esterno del controllo MultiPage.
- Per visualizzare un menu contestuale che consenta di aggiungere, rimuovere, rinominare o spostare una scheda, fate clic destro su una scheda sul controllo MultiPage.
- In fase di progettazione, per attivare una pagina fate clic sulla sua scheda. Dopo l'attivazione, potrete aggiungere alla pagina altri controlli, utilizzando le normali routine.
- Per selezionare il controllo MultiPage (anziché una pagina del controllo), fate clic sul bordo del controllo MultiPage. Tenete d'occhio la finestra Proprietà, che mostra il nome e il tipo del controllo selezionato. Potete anche selezionare il controllo MultiPage selezionandone il nome dall'elenco a discesa della finestra Proprietà,.
- Potete modificare l'aspetto del controllo MultiPage modificando le proprietà Style e TabOrientation.
- La proprietà Value di un controllo MultiPage indica quale pagina viene visualizzata. Per esempio, se scrivete del codice per impostare la proprietà Value a 0, verrà visualizzata la prima pagina del controllo MultiPage.

CONSIGLIO

Questo esempio è disponibile, in originale inglese, sul sito web del libro.

Visualizzare un grafico in una UserForm

Se avete bisogno di visualizzare un grafico in una UserForm, scoprirete che Excel non fornisce alcun modo diretto per farlo. Pertanto, dovrete essere creativi. Questo paragrafo descrive una tecnica che consente di visualizzare uno o più grafici in una UserForm.

La Figura 18.12 presenta un esempio che mostra tre grafici. La UserForm ha un controllo Image. Il trucco consiste nell'utilizzare del codice VBA per salvare il grafico come un file GIF e poi specificare tale file come proprietà Picture del controllo Image (che carica l'immagine dal disco). I pulsanti Precedente e Successivo cambiano il grafico visualizzato.

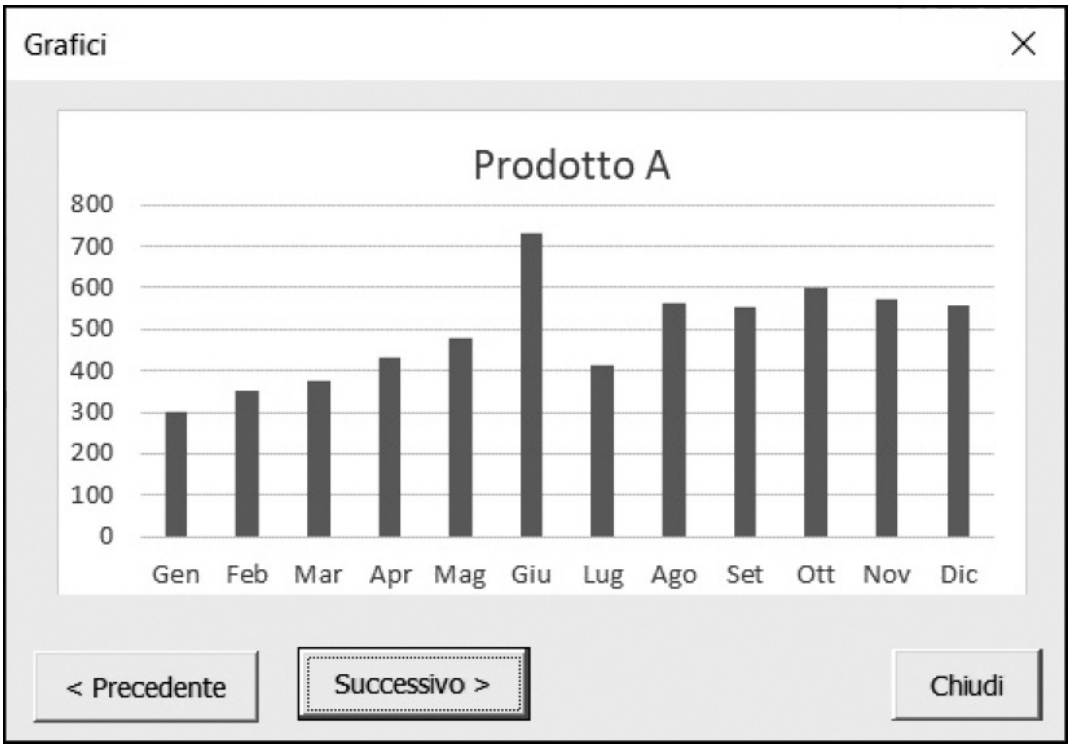

FIGURA 18.12 Visualizzazione di un grafico in una UserForm.

CONSIGLIO

In questo esempio, disponibile anche nel sito web di questo libro, i tre grafici si trovano su un foglio denominato Grafici. I pulsanti Precedente e Successivo determinano quale grafico visualizzare e il numero del grafico è memorizzato nella variabile pubblica ChartNum, accessibile da tutte le routine. La routine UpdateChart, illustrata qui, svolge tutto il lavoro:

```
Private Sub UpdateChart()
     Dim CurrentChart As Chart
     Dim Fname As String
     Set CurrentChart = _
       Sheets("Grafici").ChartObjects(ChartNum).Chart
     CurrentChart.Parent.Width = 300
     CurrentChart.Parent.Height = 150

'   Salva il grafico in formato GIF
     Fname = ThisWorkbook.Path & "\temp.gif"
     CurrentChart.Export FileName:=Fname, FilterName:="GIF"

'   Mostra il grafico
```

```
        Image1.Picture = LoadPicture(Fname)
  End Sub
```

Questa routine crea un nome per il grafico salvato e poi utilizza il metodo Export per esportare il file GIF. Infine utilizza la funzione VBA LoadPicture per specificare la proprietà Picture dell'oggetto Image.

Una checklist per le finestre di dialogo

Una finestra di dialogo è essenzialmente l'unico modo per comunicare con l'utente. Dopo tutto, le finestre di dialogo sono elementi fondamentali dell'interfaccia utente.

Quando iniziate a progettare l'interfaccia utente, eseguite i controlli presenti nel seguente elenco per assicurarvi di creare finestre di dialogo funzionali e intuitive.

- I controlli sono allineati tra loro?
- I controlli simili hanno le stesse dimensioni?
- I controlli sono equidistanti?
- La finestra di dialogo ha una didascalia appropriata?
- La finestra di dialogo contiene troppi controlli? In tal caso, potreste voler utilizzare più finestre di dialogo o suddividere i controlli con un controllo MultiPage.
- L'utente può accedere con un tasto di scelta rapida a tutti i controlli?
- Esistono duplicazioni nei tasti di scelta rapida?
- I controlli sono raggruppati in modo logico, per funzione svolta?
- L'ordine di tabulazione è impostato correttamente? L'utente dovrebbe essere in grado di attraversare la finestra di dialogo e accedere ai suoi controlli in sequenza.
- Se prevedete di memorizzare la finestra di dialogo in un componente aggiuntivo, l'avete provata a fondo dopo aver creato il componente aggiuntivo?
- Il codice VBA eseguirà l'azione appropriata se l'utente esce dalla finestra di dialogo premendo Esc o usando il pulsante Chiudi?
- Il testo contiene errori di ortografia? Sfortunatamente, il correttore ortografico di Excel non funziona con le UserForm, quindi non avrete alcun aiuto.
- La finestra di dialogo si adatta a schermi a bassa risoluzione (in genere, la modalità 1024 × 768)? In altre parole, se sviluppate la finestra di dialogo utilizzando una modalità video ad alta risoluzione, potrebbe risultare troppo grande per adattarsi a uno schermo a bassa risoluzione.
- Tutti i controlli TextBox hanno le impostazioni di convalida appropriate?
- Se intendete utilizzare la proprietà WordWrap, la proprietà MultiLine è impostata a True?
- Tutte le ScrollBar e gli SpinButtons consentono di impostare solo valori validi?
- Tutte le ListBox hanno la loro proprietà MultiSelect impostata correttamente?

Iniziate in modo semplice e sperimentate i controlli e le loro proprietà. E non dimenticate la Guida; è la fonte migliore di approfondimento di ogni controllo e proprietà.

Capitolo 19

Richiamare le macro dall'interfaccia utente

IN QUESTO CAPITOLO

- **Personalizzare la barra multifunzione**
- **Aggiungere icone alla barra di accesso rapido**
- **Modificare i menu contestuali**

Prima di Office 2007, la barra multifunzione, semplicemente, non esisteva. Allora Excel e Office usavano ancora i menu tradizionali, con le barre degli strumenti. Ora la barra multifunzione è l'interfaccia utente standard di Microsoft Office e la "moda" della barra multifunzione si è perfino estesa ad altri software per Windows.

Potreste pensare di poter creare con VBA nuovi comandi personalizzati per la barra multifunzione. La cattiva notizia è che non potete utilizzare VBA per modificare la barra multifunzione. Quella buona è che non tutto è perduto. Questo capitolo descrive alcuni dei modi per lavorare con l'interfaccia utente di Excel.

Personalizzare la barra multifunzione

Innanzitutto vedremo come si personalizza la barra multifunzione. Potete modificarla manualmente, ma non potete apportarvi modifiche con VBA. Questa è la triste verità. Se, per esempio, scrivete un'applicazione e desiderate aggiungere nuovi pulsanti alla barra multifunzione, dovrete programmare tali modifiche all'esterno di Excel, utilizzando qualcosa chiamato RibbonX.

Personalizzare manualmente la barra multifunzione

Apportare modifiche manuali alla barra multifunzione è molto semplice, ma è necessario utilizzare Excel versione 2010 o successiva. Se avete Excel 2007, saltate pure questo paragrafo, perché nel vostro caso non potrà esservi utile.

Potete personalizzare la barra multifunzione in questi modi.

- Schede
  - Aggiungere una nuova scheda personalizzata.
  - Eliminare le schede personalizzate.
  - Aggiungere un nuovo gruppo a una scheda.
  - Cambiare l'ordine delle schede.
  - Cambiare il nome di una scheda.
  - Nascondere le schede standard.
- Gruppi
  - Aggiungere nuovi gruppi personalizzati.
  - Aggiungere comandi a un gruppo personalizzato.
  - Rimuovere comandi da un gruppo personalizzato.
  - Rimuovere gruppi da una scheda.
  - Spostare un gruppo in un'altra scheda.
  - Cambiare l'ordine dei gruppi all'interno di una scheda.
  - Modificare il nome di un gruppo.

Questo è un elenco abbastanza completo delle opzioni di personalizzazione, ma ci sono alcune azioni che non si possono proprio eseguire.

- » Rimuovere le schede standard (ma potete nasconderle).
- » Rimuovere comandi dai gruppi standard.
- » Modificare l'ordine dei comandi in un gruppo standard.

Per apportare modifiche manuali alla barra multifunzione usate la scheda Barra multifunzione della finestra di dialogo Opzioni di Excel (vedi Figura 19.1). Il modo più rapido per visualizzare questa finestra di dialogo è fare clic destro in qualsiasi punto della barra multifunzione e scegliere dal menu contestuale l'opzione Personalizza barra multifunzione.

Il processo di personalizzazione della barra multifunzione è molto simile alla personalizzazione della barra di accesso rapido, descritta più avanti in questo stesso capitolo. L'unica differenza è che è necessario decidere dove inserire il comando all'interno della barra multifunzione. Seguite questa procedura generale.

1. **Utilizzate l'elenco a discesa a sinistra (Scegli comandi da) per visualizzare i vari gruppi di comandi.**
2. **Individuate il comando nel riquadro a sinistra e selezionatelo.**
3. **Utilizzate il riquadro a destra (Personalizza barra multifunzione) per scegliere un gruppo di schede.**

   Le Schede principali sono le schede sempre visibili; le Schede strumenti sono le schede contestuali che appaiono quando viene selezionato un particolare oggetto.
4. **Nel riquadro a destra, selezionate la scheda e il gruppo in cui desiderate inserire il comando.**

   Dovrete fare clic sui controlli col segno + per espandere gli elenchi gerarchici.

FIGURA 19.1 La scheda Personalizzazione barra multifunzione della finestra di dialogo Opzioni di Excel.

5. **Fate clic sul pulsante Aggiungi, nella parte centrale, per aggiungere il comando selezionato a sinistra al gruppo scelto a destra.**

Potete fare clic sul pulsante Nuova scheda per creare una nuova scheda, e sul pulsante Nuovo gruppo per creare un nuovo gruppo all'interno di una scheda. Le nuove schede e i nuovi gruppi riceveranno un nome generico, quindi probabilmente vorrete assegnare loro un nome più significativo. Per rinominare la scheda o il gruppo selezionato, utilizzate il pulsante Rinomina. Potete anche rinominare le schede e i gruppi standard.

La Figura 19.2 mostra un gruppo personalizzato, denominato Lettura vocale, aggiunto alla scheda Visualizza (a destra del gruppo Zoom). Questo nuovo gruppo offre quattro comandi.

FIGURA 19.2 La scheda Visualizza con un nuovo gruppo denominato Lettura vocale.

RICORDA!

Sebbene non sia possibile rimuovere una scheda standard, potete nasconderla deselezionando la casella di controllo accanto al suo nome nella pagina Personalizzazione barra multifunzione della finestra di dialogo Opzioni di Excel.

Aggiungere una macro alla barra multifunzione

Fortunatamente, potete anche aggiungere delle macro alla barra multifunzione. Seguite le istruzioni del paragrafo precedente ma, al passaggio 1, selezionate dall'elenco a discesa a sinistra l'opzione Macro. Saranno elencate tutte le macro attualmente disponibili, pronte per essere aggiunte alla barra multifunzione. Dovete solo decidere la scheda e un gruppo per la macro.

Se personalizzate la barra multifunzione per includervi una macro, il comando della macro sarà visibile nella barra multifunzione anche quando la cartella di lavoro che la contiene non è aperta. Facendo clic sul comando aprirete la cartella di lavoro che contiene la macro e poi eseguirete la macro.

RICORDA!

Se alla barra multifunzione aggiungete un pulsante che esegue una macro, tale modifica riguarderà solo la vostra copia di Excel. Le modifiche alla barra multifunzione non entrano a far parte della cartella di lavoro. In altre parole, se darete la vostra cartella di lavoro a un collega, le personalizzazioni della vostra barra multifunzione non appariranno sul sistema del collega.

Personalizzare la barra multifunzione con il linguaggio XML

In alcune situazioni, potreste voler far sì che la barra multifunzione si modifichi automaticamente quando viene aperta una cartella di lavoro o un componente aggiuntivo. In questo modo sarà più facile per l'utente accedere alla vostra macro. Elimina inoltre la necessità per l'utente finale di modificare manualmente la barra multifunzione passando dalla finestra di dialogo Opzioni di Excel.

Potete apportare modifiche automatiche alla barra multifunzione con Excel 2007 e versioni successive, ma non è un'operazione agevole. La modifica della barra multifunzione comporta la scrittura di codice XML in un editor di testi, la copia del file XML nel file della cartella di lavoro, la modifica di alcuni file XML (anch'essi archiviati all'interno del file Excel, che in realtà altro non è che uno zip di singoli file) e infine la scrittura delle routine VBA necessarie per gestire il clic sui controlli inseriti nel file XML.

Fortunatamente, per aiutarvi a personalizzare la barra multifunzione è disponibile del software, ma il problema è che dovrete comunque avere una certa familiarità con il linguaggio XML.

SCARICARE IL SOFTWARE

Se intendete seguire l'esempio di personalizzazione della barra multifunzione, dovete scaricare *Custom UI Editor*, un piccolo programma per Microsoft Office. Questo programma gratuito semplifica enormemente il processo di personalizzazione della barra multifunzione nelle applicazioni per Microsoft Office. Pur con questo software, avrete bisogno ancora di fare molto lavoro, ma è sempre molto più semplice che fare tutto manualmente.

La pagina da cui eseguire il download tende a cambiare, quindi per trovare il software cercate nel Web "Custom UI Editor Microsoft Office". È un download piccolo e gratuito. Spiegare tutti i dettagli per la personalizzazione

della barra multifunzione esula dagli scopi di questo libro. Tuttavia, in questo capitolo presento un breve esempio che illustra i passaggi necessari per aggiungere un nuovo gruppo alla scheda Home della barra multifunzione. Il nuovo gruppo, denominato Excel VBA For Dummies, contiene un pulsante, denominato Click Me. Facendo clic su questo pulsante viene eseguita una macro VBA, denominata ShowMessage.

CONSIGLIO

Potete scaricare un file di esempio, in inglese, dal sito web di questo libro, che contiene questa personalizzazione. Se desiderate eseguirla voi stessi, seguite con precisione questi passaggi.

1. **Create una nuova cartella di lavoro di Excel.**
2. **Salvatela e denominatela ribbon modification.xlsm.**
3. **Chiudete la cartella di lavoro.**
4. **Avviate Custom UI Editor for Microsoft Office.**

   Se non disponete di questo software, trovatelo e installatelo. Fate riferimento al riquadro “Scaricare il software”.
5. **In Custom UI Editor, selezionate il comando File ➪ Open e trovate la cartella di lavoro salvata nel passaggio 2.**
6. **Scegliete il comando Insert ➪ Office 2007 Custom UI Part.**

   Scegliete questo comando anche se utilizzate Excel versione 2010 o successiva.
7. **Digitate il seguente codice nel riquadro del codice (denominato customUI.xml) visualizzato in Custom UI Editor (vedi la Figura 19.3):**

```
<customUI xmlns='http://schemas.microsoft.com/office/2006/01/customui'>
<ribbon>
<tabs>
<tab idMso='TabHome'>
     <group id'Group1' label='Excel VBA For Dummies'>
       <button id='Button1'
          label='Click Me'
          size='large'
          onAction='ShowMessage'
          imageMso='FileStartWorkflow'/>
       </group>
</tab>
</tabs>
</ribbon>
</customUI>
```

FIGURA 19.3 Codice RibbonX visualizzato in Custom UI Editor.

8. **Fate clic sul pulsante Validate sulla barra degli strumenti.**

   Se il codice presenta errori di sintassi, verrà visualizzato un messaggio che descrive il problema. Se vengono identificati errori, dovrete correggerli.

9. **Fate clic sul pulsante Generate Callback.**

   Custom UI Editor crea una routine Sub che viene eseguita quando fate clic sul pulsante (vedi la Figura 19.4). Questa routine non viene inserita nella cartella di lavoro, quindi dovrete copiarla, per poterla poi utilizzare.

FIGURA 19.4 La routine VBA che viene eseguita facendo clic sul pulsante nella barra multifunzione.

10. **Tornate al modulo customUI.xml e selezionate il comando File ➪ Save (o fate clic sull'icona Save sulla barra degli strumenti).**

11. **Chiudete il file scegliendo File ➪ Close.**

12. **Aprite la cartella di lavoro in Excel e fate clic sulla scheda Home.**

    Dovreste vedere il nuovo gruppo e il pulsante nella barra multifunzione. Ma il tutto non è ancora funzionante.

13. **Premete Alt + F11 per richiamare il VBE.**

14. **Inserite un nuovo modulo VBA; incollate (o digitate) la routine Callback generata nel Passo 9; aggiungete un'istruzione MsgBox: in tal modo saprete se la routine è effettivamente in esecuzione.**

    La routine è

```
Sub ShowMessage(control As IRibbonControl)
     MsgBox "Avete un nuovo comando sulla barra."
End Sub
```

15. **Premete Alt + F11 per tornare a Excel e fate clic sul nuovo pulsante sulla barra multifunzione.**

    Se tutto va bene, vedrete la MsgBox rappresentata nella Figura 19.5.

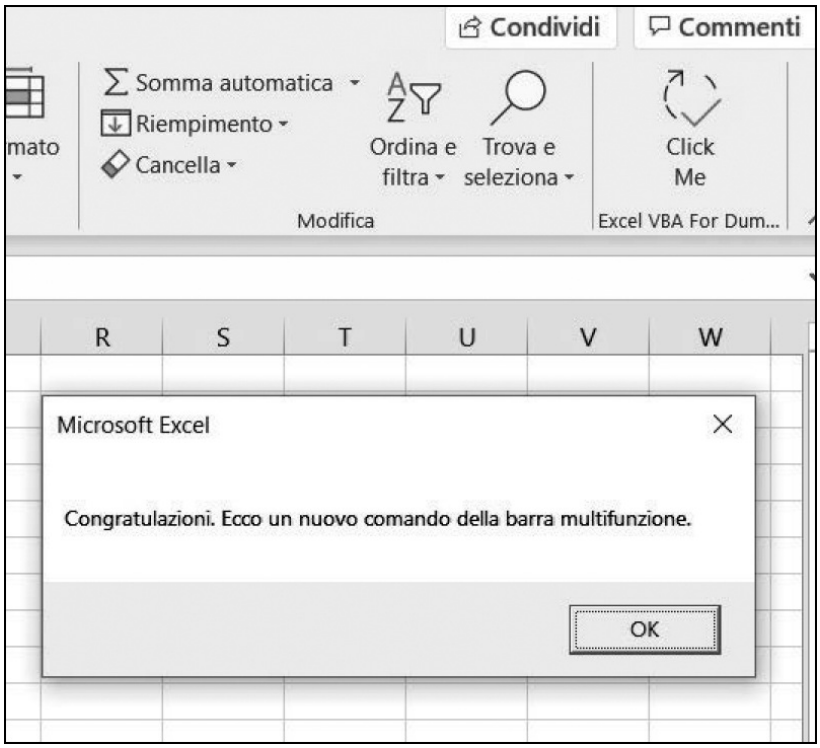

FIGURA 19.5 La dimostrazione che l'aggiunta di un nuovo comando tramite XML è effettivamente possibile.

RICORDA!

In Custom UI Editor, quando selezionate Insert ➪ Office 2007 Custom UI Part, inserite un elemento per Excel 2007. L'editor dell'interfaccia utente personalizzata ha anche la possibilità di inserire un elemento per Excel 2010 (il software non è stato aggiornato per le versioni successive di Office). Per la massima compatibilità, utilizzate però l'opzione Excel 2007 Custom UI Part.

Vi sarete probabilmente resi conto che modificare la barra multifunzione con XML non è esattamente una passeggiata. Anche con un buon strumento di supporto (come Custom UI Editor), occorre conoscere il linguaggio XML. Se la cosa vi sembra interessante, cercate istruzioni nel Web o trovate un libro dedicato esclusivamente alla personalizzazione dell'interfaccia a barra multifunzione di Microsoft Office. Purtroppo, in questo libro non posso occuparmene più di tanto.

Poiché la programmazione XML è un argomento troppo complesso per un programmatore VBA alle prime armi, il resto di questo capitolo si concentra sulla personalizzazione dell'interfaccia utente utilizzando il vecchio metodo (solo VBA): in particolare, vedremo come personalizzare i menu contestuali. Non sarà uno strumento accessibile come la barra multifunzione, ma è molto più semplice e offre comunque un rapido accesso alle macro.

AGGIUNGERE UN NUOVO PULSANTE ALLA BARRA DI ACCESSO RAPIDO

Se create una macro che pensate di utilizzare frequentemente, potete aggiungere un nuovo pulsante alla barra di accesso rapido. È facile, ma dovete procedere manualmente. La barra di accesso rapido può essere personalizzata solo dall'utente finale, non da programma. Ecco come procedere.

1. **Fate clic destro sulla barra di accesso rapido e selezionate dal menu contestuale il comando Personalizza barra di accesso rapido per visualizzare la scheda Barra di accesso rapido della finestra di dialogo Opzioni di Excel.**
2. **Dall'elenco a discesa Scegli comandi da, selezionate Macro.**
3. **Selezionate la vostra macro dall'elenco.**
4. **Fate clic sul pulsante Aggiungi per aggiungere la macro all'elenco della barra di accesso rapido, sul lato destro.**
5. **Se lo desiderate, fate clic sul pulsante Modifica per modificare l'icona e (facoltativamente) il nome visualizzato.**

Quando farete clic su un pulsante per macro della Barra di accesso rapido, si aprirà la cartella di lavoro contenente la macro (se non è già aperta). La macro può essere eseguita solo quando la sua cartella di lavoro è aperta.

Troverete anche un'opzione per visualizzare il pulsante della barra di accesso rapido solo quando è aperta una determinata cartella di lavoro. Prima di aggiungere la macro, utilizzate il controllo a discesa nella parte superiore destra della finestra di dialogo Opzioni di Excel e specificate il nome della cartella di lavoro anziché Per tutti i documenti (impostazione predefinita). Se avete delle macro utili per più cartelle di lavoro, può essere una buona idea salvarle nella vostra Cartella macro personale.

Personalizzare i menu contestuali

Prima di Excel 2007, i programmatori VBA utilizzavano l'oggetto CommandBar per creare menu, barre degli strumenti personalizzate e menu contestuali (richiamati col tasto destro del mouse) personalizzati.

A partire da Excel 2007, l'oggetto CommandBar si trova in una posizione piuttosto strana. Se scrivete del codice per personalizzare un menu o una barra degli strumenti, Excel intercetta quel codice e ignora molti dei vostri comandi. Invece di mostrare i vostri perfezionamenti dell'interfaccia, Excel 2007 (come le versioni successive) relega i menu e le barre degli strumenti personalizzati in una scheda "passepartout" della barra multifunzione denominata Componenti aggiuntivi.

Le personalizzazioni dei menu e delle barre degli strumenti finiscono nel gruppo Componenti aggiuntivi ➪ Comandi di menu o Componenti aggiuntivi ➪ Barre degli strumenti personalizzate. Ma la personalizzazione dei menu contestuali (che utilizza anch'essa l'oggetto CommandBar) funziona ancora "come si deve". Be'… più o meno. Vedete il paragrafo "Che cosa cambia rispetto a Excel 2007?", più avanti in questo capitolo.

E quindi? Quindi l'oggetto CommandBar non è più molto utile, ma rimane l'unico modo per personalizzare i menu contestuali.

Aggiungere un nuovo elemento al menu contestuale della cella

Questo paragrafo contiene del codice di esempio che aggiunge un nuovo elemento al menu contestuale visualizzato quando fate clic destro su una cella. Sebbene i dettagli tecnici non rientrino negli ambiti di questo libro, dovreste essere in grado di adattare questi esempi alle vostre esigenze.

Il Capitolo 16 descrive l'utility Maiuscole e minuscole. Potete potenziarla un po' rendendola disponibile dal menu contestuale della cella.

CONSIGLIO

Questo esempio è disponibile, in originale inglese, sul sito web di questo libro.

La routine AddToShortcut aggiunge una nuova voce al menu contestuale della cella. Potete adattarla per farla puntare alle vostre macro modificando le proprietà Caption e OnAction dell'oggetto NewControl.

```
Sub AddToShortCut()
     Dim Bar As CommandBar
     Dim NewControl As CommandBarButton
     DeleteFromShortcut
     Set Bar = Application.CommandBars("Cell")
     Set NewControl = Bar.Controls.Add _
         (Type:=msoControlButton, ID:=1, _
         temporary:=True)
     With NewControl
         .Caption = "&Maiuscole/minuscole"
         .OnAction = "ChangeCase"
         .Style = msoButtonIconAndCaption
     End With
End Sub
```

RICORDA!

Quando si modifica un menu contestuale, quella modifica rimane in vigore finché non si riavvia Excel. In altre parole, i menu contestuali modificati non vengono ripristinati quando si chiude la cartella di lavoro che contiene il codice VBA. Pertanto, se scrivete del codice per modificare un menu contestuale, quasi sempre dovrete scrivere anche del codice per invertire l'effetto della modifica.

La routine DeleteFromShortcut rimuove la nuova voce di menu dal menu contestuale della cella:

```
Sub DeleteFromShortcut()
     On Error Resume Next
     Application.CommandBars("Cell").Controls _
         ("&Maiuscole/minuscole").Delete
End Sub
```

La Figura 19.6 mostra la nuova voce di menu visualizzata dopo aver fatto clic destro su una cella.

Il primo vero comando dopo la dichiarazione di un paio di variabili richiama la routine DeleteFromShortcut. Questa istruzione garantisce che nel menu contestuale della cella venga visualizzata una sola voce Maiuscole/minuscole. Provate a trasformare in commento quella riga (mettete un apostrofo all'inizio della riga) ed eseguite la routine alcune volte (ma non lasciatevi trasportare!). Fate clic destro su una cella e potrete visualizzare più istanze della voce di menu Maiuscole/minuscole. Sbarazzatevi di tutte le voci eseguendo più volte DeleteFromShortcut (una volta per ogni voce di menu di troppo).

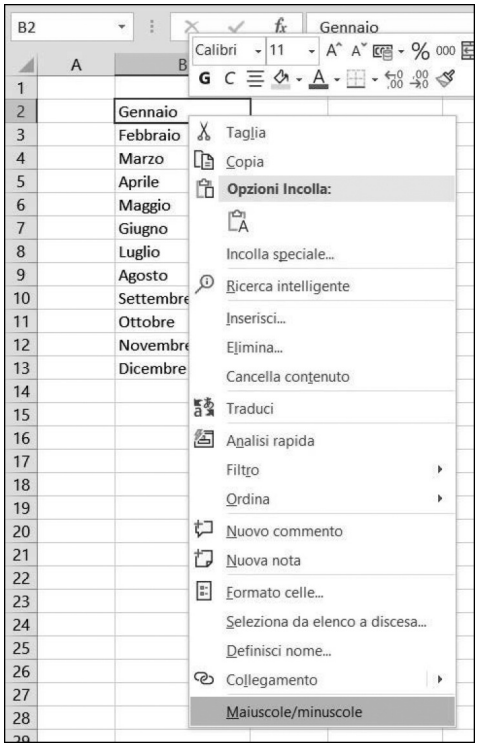

FIGURA 19.6 Il menu contestuale della cella mostra una voce di menu personalizzata: Maiuscole/minuscole.

Infine, è necessario un modo per aggiungere la voce del menu contestuale quando viene aperta la cartella di lavoro e poi per eliminarla quando la cartella di lavoro viene chiusa. Farlo è facile... se avete letto il Capitolo 11. Basta aggiungere queste due routine per l'evento al modulo di codice Questa_cartella_di_lavoro:

```
Private Sub Workbook_Open()
     Call AddToShortCut
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
     Call DeleteFromShortcut
End Sub
```

La routine Workbook_Open viene eseguita quando la cartella di lavoro viene aperta e la routine Workbook_BeforeClose viene eseguita prima che la cartella di lavoro venga chiusa. Proprio quello che ha ordinato il dottore.

Che cosa cambia rispetto a Excel 2007?

Se avete già utilizzato VBA per lavorare sui menu contestuali di Excel 2007 o versioni precedenti, occorre essere consapevoli di un cambiamento significativo.

In passato, se il codice modificava un menu contestuale, quella modifica valeva per tutte le cartelle di lavoro. Per esempio, se avevate aggiunto un nuovo elemento al menu contestuale della cella, tale nuovo elemento veniva visualizzato per un clic destro su una cella di quella cartella di lavoro e di tutte le altre cartelle di lavoro che potevate aprire in seguito. In altre parole, le modifiche al menu contestuale valevano a livello dell'intera applicazione.

Excel versione 2013 e successive utilizzano un'interfaccia mono-documento e ciò ha effetto sui menu contestuali. Le modifiche apportate ai menu contestuali riguardano solo la finestra della cartella di lavoro attiva. Quando si esegue il codice che modifica il menu contestuale, il menu contestuale di altre finestre, diverse dalla finestra attiva, non viene modificato. Questo rappresenta un cambiamento radicale rispetto al modo in cui le cose funzionavano prima.

Un'altra differenza: se l'utente apre una cartella di lavoro (o crea una nuova cartella di lavoro) quando la finestra attiva è dotata del menu contestuale modificato, anche la nuova cartella di lavoro erediterà il menu contestuale modificato. In altre parole, aprendo nuove finestre, queste adotteranno gli stessi menu contestuali della finestra attiva.

In conclusione: in passato, se si apriva una cartella di lavoro o un componente aggiuntivo che modificava i menu contestuali, si poteva essere certi che i menu contestuali modificati sarebbero stati disponibili in tutte le cartelle di lavoro. Ora non avete più questa certezza.

5

Riepilogando...

IN QUESTA PARTE...

- Perché può essere utile creare funzioni personalizzate per il foglio di lavoro.
- Utilizzare al meglio le nuove funzioni personalizzate esattamente come le funzioni standard di Excel.
- I componenti aggiuntivi di Excel.
- Creare semplici componenti aggiuntivi.

Capitolo 20

Creare nuove funzioni personalizzate per il foglio di lavoro

IN QUESTO CAPITOLO

- **Perché le funzioni personalizzate possono essere così utili in un foglio di lavoro**
- **Le basi delle funzioni personalizzate del foglio di lavoro**
- **Scrivere nuove funzioni**
- **Funzioni che utilizzano vari tipi di argomenti**
- **Esempi di funzioni**
- **La finestra di dialogo Inserisci funzione**

Per molti esperti di macro, il vero punto di forza del linguaggio VBA è la capacità di creare funzioni personalizzate per il foglio di lavoro, funzioni che hanno l'aspetto e che operano esattamente come quelle standard di Excel. Una *funzione personalizzata* offre il vantaggio aggiuntivo di lavorare esattamente come desiderate (perché l'avete scritta voi stessi). Nel Capitolo 5 ho già introdotto il concetto di funzioni personalizzate. In questo capitolo ho predisposto un'immersione più in profondità, con alcuni esempi tratti dal mondo reale.

Perché creare funzioni personalizzate?

Avete sicuramente familiarità con le funzioni del foglio di lavoro di Excel; anche i neofiti di Excel sanno usare le più comuni funzioni del foglio di lavoro, come SOMMA (SUM), MEDIA (AVERAGE) e SE (IF). Excel offre oltre 450 funzioni standard, ma se ciò non fosse sufficiente, potete creare nuove funzioni utilizzando il linguaggio VBA.

Con tutte queste funzioni disponibili in Excel e VBA, potreste chiedervi perché mai dovreste aver bisogno di creare altre funzioni. La risposta: per semplificarvi il lavoro. Con un minimo di pianificazione, le funzioni personalizzate possono essere molto utili nelle formule del foglio di lavoro e nelle routine VBA. Spesso, per fare un esempio, potete abbreviare notevolmente una formula, creando una funzione personalizzata. Dopo tutto, se le formule sono più brevi saranno anche più comprensibili e più facili da utilizzare.

CHE COSA NON POSSONO FARE LE FUNZIONI PERSONALIZZATE

Nello sviluppo di funzioni personalizzate da utilizzare nelle formule di un foglio di lavoro, è importante comprendere un punto chiave. Le routine VBA Function sono essenzialmente passive. Per esempio, il codice all'interno di una routine Function non può modificare gli intervalli, modificare la formattazione o eseguire molte delle altre operazioni che invece sono possibili con una routine Sub. Un esempio chiarirà questo concetto.

Sarebbe bello poter creare una funzione che cambi il colore del testo contenuto in una cella sulla base del valore contenuto nella cella stessa. Ma non si può scrivere una tale funzione: vi restituirà sempre un valore di errore.

Basti ricordare questo: una funzione utilizzata in una formula del foglio di lavoro non fa altro che calcolare e restituire un valore. Non può eseguire azioni sugli oggetti.

Detto questo, questa ferrea regola ha alcune eccezioni. Per esempio, ecco una routine Function che cambia il testo del commento di una cella:

```
Function ChangeComment(cell, NewText)
    cell.Comment.Text NewText
End Function
```

Ed ecco una formula che usa la funzione. Funziona solo se la cella A1 contiene già un commento. Quando viene calcolata la formula, il commento viene modificato.

```
=ChangeComment(A1,"Ho cambiato il commento!")
```

Non è chiaro se si tratti di una svista o di una funzionalità decisa. Ma è un raro esempio di una funzione VBA che riesce a modificare qualcosa in un foglio di lavoro.

Concetti di base sulle funzioni VBA

Una *funzione VBA* è una routine memorizzata in un modulo VBA. Potete utilizzare queste funzioni in altre routine VBA o nelle formule del foglio di lavoro. Le funzioni personalizzate non possono essere create con il registratore di macro, sebbene il registratore di macro possa aiutare a identificare proprietà e metodi utili.

Un *modulo* può contenere un numero qualsiasi di funzioni. In una formula potete anche utilizzare una nuova funzione personalizzata come se fosse una funzione standard. Se però avete definito la funzione in un'altra cartella di lavoro, dovrete far precedere al nome della funzione il nome della cartella di lavoro. Supponiamo di sviluppare una funzione chiamata DiscountPrice (che accetta un argomento) e archiviata nella cartella di lavoro pricing.xlsm.

Per utilizzare questa funzione nella cartella di lavoro pricing.xlsm, potete usare una formula come questa:

```
=DiscountPrice(A1)
```

Se però dovete utilizzare questa funzione in un'altra cartella di lavoro, dovrete specificare una formula come la seguente (e assicuratevi anche che il file pricing. xlsm sia aperto):

```
=pricing.xlsm!discountprice(A1)
```

CONSIGLIO

Se la funzione personalizzata è memorizzata in un componente aggiuntivo, non è necessario far precedere al nome della funzione il nome della cartella di lavoro. Il Capitolo 21 offre una panoramica dei componenti aggiuntivi.

Le funzioni personalizzate vengono anche presentate nella finestra di dialogo Inserisci funzione, all'interno della categoria Definite dall'utente. Un modo per richiamare la finestra di dialogo Inserisci funzione consiste nel premere la combinazione di tasti Maiusc + F3.

Scrivere nuove funzioni

Ricordate sempre che il nome di una funzione si comporta come una variabile. Il valore finale di questa variabile sarà poi il valore restituito dalla funzione. Per mostrare questo concetto in azione, esaminiamo la seguente funzione, che restituisce il nome dell'utente:

```
Function FirstName()
     Dim FullName As String
     Dim FirstSpace As Long
     FullName = Application.UserName
     FirstSpace = InStr(FullName, " ")
     If FirstSpace = 0 Then
         FirstName = FullName
     Else
         FirstName = Left(FullName, FirstSpace - 1)
     End If
End Function
```

Questa funzione inizia assegnando alla variabile FullName la proprietà UserName dell'oggetto Application. Successivamente, utilizza la funzione VBA InStr per individuare la posizione del primo spazio all'interno del nome. Se non c'è alcuno spazio, FirstSpace è uguale a 0 e FirstName è uguale all'intero nome. Se FullName ha uno spazio, la funzione Left estrae il testo alla sinistra dello spazio e lo assegna a FirstName.

Notate che FirstName è il nome della funzione e, nella funzione, viene anche utilizzato come nome di variabile. Il valore finale di FirstName verrà poi restitui to dalla funzione. Nella funzione possono essere svolti diversi calcoli intermedi, ma la funzione restituisce sempre l'ultimo valore assegnato alla variabile che ha lo stesso nome della funzione.

CONSIGLIO

Tutti gli esempi di questo capitolo sono disponibili (in originale inglese) sul sito web di questo libro.

Gli argomenti delle funzioni

Per usare le funzioni, è necessario capire come si utilizzano i suoi argomenti. Per "argomento" non si intende un discorso mirato su un solo soggetto. Si tratta piuttosto delle informazioni che passate alla funzione, e che poi la funzione utilizzerà per svolgere i suoi calcoli.

I punti che seguono riguardano sia gli argomenti delle funzioni standard di Excel sia le funzioni VBA personalizzate.

- Gli *argomenti* possono essere riferimenti a celle, variabili (inclusi array, che Excel chiama matrici), costanti, valori letterali o espressioni.
- Alcune funzioni non hanno argomenti.
- Alcune funzioni hanno un numero fisso di argomenti obbligatori.
- Alcune funzioni hanno una combinazione di argomenti obbligatori e opzionali.

Gli esempi di questo paragrafo mostrano come lavorare con vari tipi di argomenti.

Una funzione senza argomenti

Alcune funzioni non usano argomenti. Per esempio, Excel offre alcune funzioni standard che non usano argomenti. Tra queste CASUALE (RAND), OGGI (TODAY) e ADESSO (NOW).

Ecco un esempio di funzione personalizzata senza argomenti. La funzione seguente restituisce la proprietà UserName dell'oggetto Application. Questo nome viene visualizzato nella scheda Generale della finestra di dialogo Opzioni di Excel. Questo semplice ma significativo esempio mostra l'unico modo in cui potete far apparire il nome dell'utente in una cella del foglio di lavoro:

```
Function User()
'   Restituisce il nome dell'utente corrente
     User = Application.UserName
End Function
```

Inserendo la seguente formula in una cella del foglio di lavoro, la cella visualizzerà, appunto, il nome dell'utente corrente:

```
=User()
```

Come con le funzioni standard di Excel, quando utilizzate una funzione, anche senza argomenti, è comunque necessario specificare una coppia di parentesi vuote. In caso contrario, Excel tenterà di interpretare la funzione come se fosse un intervallo dotato di nome.

Una funzione con un argomento

La funzione a singolo argomento di questo paragrafo è progettata per i responsabili commerciali che devono calcolare le commissioni guadagnate dai loro venditori. La percentuale delle commissioni dipende dal volume delle vendite mensili; chi vende di più gode di una percentuale di commissioni più favorevole. La funzione restituisce l'importo delle commissioni in base alle vendite mensili (che è l'unico argomento della funzione, un argomento obbligatorio). I calcoli di questo esempio si basano sulla Tabella 20.1.

TABELLA 20.1 Tassi di commissione per le vendite

Vendite mensili (importi)	Commissioni di vendita
Fino a 9.999	8,0%
10.000 – 19.999	10,5%
20.000 – 39.999	12,0%
Da 40.000 in su	14,0%

Per calcolare le commissioni per gli importi delle vendite inseriti in un foglio di lavoro potete utilizzare diversi approcci. Potreste scrivere nel foglio di lavoro una lunga formula, come questa:

```
=SE(E(A1>=0;A1<=9999,99);A1*0,08;
SE(E(A1>=10000;A1<=19999,99);A1*0,105;
SE(E(A1>=20000;A1<=39999,99);A1*0,12;
SE(A1>=40000;A1*0,14;0))))
```

Esistono un paio di motivi che rendono sconsigliabile questo approccio. Innanzitutto, la formula è eccessivamente complessa. In secondo luogo, i valori sono specificati direttamente all'interno della formula, rendendola difficile da modificare se il calcolo delle commissioni dovesse cambiare.

Un approccio migliore consiste nel creare una tabella di valori di commissioni e utilizzare una funzione per tabelle CERCA.VERT (VLOOKUP) per calcolare le commissioni:

```
=CERCA.VERT(A1;Tabella;2)*A1
```

Un altro approccio, che non richiede una tabella delle commissioni, consiste nel creare una funzione personalizzata:

```
Function Commission(Sales)
'   Calcola le commissioni per le vendite
     Const Tier1 As Double = 0.08
     Const Tier2 As Double = 0.105
     Const Tier3 As Double = 0.12
     Const Tier4 As Double = 0.14
     Select Case Sales
               Case 0 To 9999.99: Commission = Sales * Tier1
               Case 10000 To 19999.99: Commission = Sales * Tier2
               Case 20000 To 39999.99: Commission = Sales * Tier3
               Case Is >= 40000: Commission = Sales * Tier4
          End Select
          Commission = Round(Commission, 2)
End Function
```

Notate che le quattro percentuali di commissioni sono dichiarate all'inizio come costanti anziché essere specificate direttamente nel calcolo. Questo rende molto facile modificare la funzione quando in futuro tali percentuali dovessero variare.

Dopo aver definito questa funzione in un modulo VBA, potrete tranquillamente utilizzarla in una formula del foglio di lavoro. Inserendo la seguente formula in una cella, otterrete il risultato 3.000. L'importo delle vendite pari a 25.000 si qualifica per una percentuale di commissioni del 12%:

```
=Commission(25000)
```

La Figura 20.1 mostra un foglio di lavoro che nelle formule nella colonna C utilizza proprio questa funzione Commission.

Una funzione con due argomenti

Il prossimo esempio si basa su quello precedente. Immaginate che il responsabile commerciale implementi una nuova politica per premiare i dipendenti di maggiore esperienza: le commissioni totali aumentano dell'1% per ogni anno di anzianità in azienda del venditore.

C4 | =Commission(B4)

	A	B	C	D
1	Nome	Vendite	Commissioni	
2	Adams	61.983,00 €	8.677,62 €	
3	Baker	3.506,00 €	280,48 €	
4	Douglas	38.973,00 €	4.676,76 €	
5	Emmett	32.092,00 €	3.851,04 €	
6	Franklin	27.354,00 €	3.282,48 €	
7	Johnson	17.833,00 €	1.872,46 €	
8	Kent	41.598,00 €	5.823,72 €	
9	Mays	32.000,00 €	3.840,00 €	
10	Quincy	5.000,00 €	400,00 €	
11	Randall	68.793,00 €	9.631,02 €	
12	Smith	31.093,00 €	3.731,16 €	
13	Walker	24.509,00 €	2.941,08 €	
14	Zeller	41.544,00 €	5.816,16 €	
15				
16				

FIGURA 20.1 Uso della funzione Commission in un foglio di lavoro.

Potete modificare la funzione Commission (definita nel paragrafo precedente) in modo che utilizzi due argomenti, entrambi obbligatori. Chiameremo questa nuova funzione Commission2:

```
Function Commission2(Sales, Years)
'   Calcola le commissioni in base all'anzianità di servizio
```

```
        Const Tier1 As Double = 0.08
        Const Tier2 As Double = 0.105
        Const Tier3 As Double = 0.12
        Const Tier4 As Double = 0.14
        Select Case Sales
               Case 0 To 9999.99: Commission = Sales * Tier1
               Case 10000 To 19999.99: Commission = Sales * Tier2
               Case 20000 To 39999.99: Commission = Sales * Tier3
               Case Is >= 40000: Commission = Sales * Tier4
             End Select
             Commission2 = Commission2 + (Commission2 * Years / 100)
        Commission2 = Round(Commission2, 2)
End Function
```

La routine aggiunge semplicemente il secondo argomento (Years) all'istruzione Function e include un calcolo aggiuntivo che incrementa le commissioni prima di uscire dalla funzione. Questo calcolo aggiuntivo moltiplica le commissioni base per il numero di anni in servizio, divide il risultato per 100 e poi somma il risultato al calcolo originale.

Ecco un esempio di come potete scrivere una formula utilizzando questa funzione (si presuppone che la cella A1 contenga l'importo delle vendite e che la cella B1 specifichi il numero di anni di servizio del venditore).

```
=Commission2(A1,B1)
```

La Figura 20.2 mostra un foglio di lavoro che utilizza la funzione Commission2.

Una funzione con un argomento intervallo

Utilizzare un intervallo di fogli di lavoro come argomento non è affatto complicato; Excel si occupa dei dettagli dietro le quinte.

FIGURA 20.2 Uso della funzione Commission2, che accetta due argomenti.

Ecco una semplice ma utile funzione che concatena i dati contenuti in un intervallo. Sono necessari due argomenti: InRange (l'intervallo del foglio di lavoro da concatenare) e Delim (uno o più caratteri delimitatori, da inserire tra le celle).

```
Function JoinText(InRange, Delim)
     Dim Cell As Range
     For Each Cell In InRange
         JoinText = JoinText & Cell.Value & Delim
     Next Cell
     JoinText = Left(JoinText, Len(JoinText) - Len(Delim))
End Function
```

Per eseguire il ciclo su ciascuna cella dell'intervallo usa un costrutto For Each-Next. Concatena il contenuto della cella, seguito dai caratteri Delim. L'ultima istruzione rimuove il delimitatore finale, che non è necessario, perché non ci sono più elementi.

La Figura 20.3 mostra un esempio d'uso. Il secondo argomento è una stringa di due caratteri (una virgola seguita da uno spazio).

FIGURA 20.3 Uso della funzione JoinText per concatenare un insieme di celle.

Ecco ora un altro esempio di una funzione che utilizza un argomento che è un intervallo di celle. Supponiamo che dobbiate calcolare la media dei cinque valori maggiori di un intervallo denominato Data. Excel non ha una funzione in grado di farlo, quindi probabilmente dovreste scrivere una formula come:

```
=(GRANDE(Data;1)+GRANDE(Data;2)+GRANDE(Data;3)+
GRANDE(Data;4)+GRANDE(Data;5))/5
```

Questa formula utilizza la funzione GRANDE (LARGE) di Excel, che restituisce l'*n*-esimo valore più grande di un intervallo. La formula somma fra loro i cinque valori più grandi dell'intervallo di celle Data e poi divide il risultato per 5. La formula funziona correttamente, ma è piuttosto ingombrante. E se poi voleste calcolare la media dei primi sei valori? Dovreste riscrivere la formula e assicurarvi di aggiornare tutte le sue copie.

Non sarebbe più semplice se Excel avesse una funzione chiamata TopAvg? Quindi potreste calcolare la media utilizzando la seguente funzione:

```
=TopAvg(Data,5)
```

Questo esempio mostra un caso in cui una funzione personalizzata può semplificare le cose. La seguente funzione VBA personalizzata TopAvg, restituisce la media degli n valori più grandi di un intervallo di celle:

```
Function TopAvg(InRange, N)
'   Restituisce la media degli n valori più grandi di InRange
     Dim Sum As Double
     Dim i As Long
     Sum = 0
     For i = 1 To N
         Sum = Sum + WorksheetFunction.Large(InRange, i)
     Next i
     TopAvg = Sum / N
End Function
```

Questa funzione accetta due argomenti: InRange (che è un intervallo di fogli di lavoro) e N (il numero di valori di cui calcolare la media). Prima di tutto inizializza la variabile Sum a 0. Quindi utilizza un ciclo For-Next per calcolare la somma degli n valori più grandi dell'intervallo. Notate l'uso della funzione di Excel LARGE (GRANDE) all'interno del ciclo. Infine, alla variabile TopAvg viene assegnato il valore della somma divisa per *N*.

CONSIGLIO

Nelle vostre routine VBA potete utilizzare tutte le funzioni del foglio di lavoro di Excel (rigorosamente specificate in inglese), tranne quelle che hanno una funzione equivalente in VBA. Per esempio, in VBA esiste una funzione Rnd che restituisce un numero casuale. Pertanto, in una routine VBA non potete utilizzare la funzione RAND (CASUALE) di Excel.

Una funzione con un argomento opzionale

Molte funzioni standard di Excel utilizzano argomenti facoltativi. Un esempio è la funzione SINISTRA (LEFT), che restituisce i caratteri tratti dal lato sinistro di una stringa. La sua sintassi è la seguente:

```
=SINISTRA(testo[,numero_caratteri])
```

Il primo argomento è obbligatorio, ma il secondo (specificato tra parentesi quadre) è facoltativo. Se omettete l'argomento opzionale, Excel usa automaticamente il valore 1. Pertanto, le seguenti formule restituiscono lo stesso risultato:

```
=SINISTRA(A1,1)
=SINISTRA(A1)
```

Anche le funzioni personalizzate sviluppate in VBA possono avere argomenti opzionali. Potete specificare che un argomento è opzionale facendo precedere al nome dell'argomento la parola riservata Optional. Il nome dell'argomento dev'essere poi seguito da un segno di uguale e dal valore predefinito. Quando non specificate l'argomento opzionale, il codice utilizza il valore predefinito.

RICORDA!

Un avvertimento: gli eventuali argomenti opzionali, devono essere specificati solo dopo tutti gli argomenti obbligatori nell'elenco degli argomenti.

L'esempio seguente mostra una funzione personalizzata che utilizza un argomento opzionale:

```
Function DrawOne(InRange, Optional Recalc = 0)
      '   Sceglie casualmente una cella tratta da un intervallo

      Randomize
'   Rende volatile la funzione se Recalc = 1
      If Recalc = 1 Then Application.Volatile True

'   Determina una cella casuale
      DrawOne = InRange(Int((InRange.Count) * Rnd + 1))
End Function
```

Questa funzione sceglie casualmente una cella da un intervallo di input. L'intervallo passato come argomento è in realtà una matrice e la funzione seleziona casualmente un elemento dalla matrice (per un aggiornamento sulle matrici consultate il Capitolo 7). Se il secondo argomento è 1, il valore selezionato viene modificato ogni volta che il foglio di lavoro viene ricalcolato (la funzione è volatile). Se il secondo argomento è 0 (o viene omesso), la funzione non viene ricalcolata, a meno che non sia stata modificata una delle celle dell'intervallo di input.

L'istruzione Randomize garantisce che venga scelto un numero casuale (seme) differente ogni volta che viene aperta la cartella di lavoro. Senza questa istruzione, a ogni apertura della cartella di lavoro verranno generati sempre gli stessi numeri casuali.

Potete utilizzare questa funzione per darvi i numeri della lotteria, per selezionare un vincitore a caso da un elenco di nomi e così via.

DEBUG DELLE FUNZIONI PERSONALIZZATE

Il debug di una routine Function può essere più impegnativo rispetto a quello di una routine Sub. Se sviluppate una funzione da utilizzare poi nelle formule del foglio di lavoro, scoprirete che un errore nella routine Function si traduce semplicemente in una visualizzazione di errore nella cella della formula (in genere, #VALORE!). In altre parole, non si riceve il normale messaggio d'errore runtime che consentirebbe di individuare l'istruzione incriminata.

Per il debug delle funzioni personalizzate potete scegliere fra tre metodi.

- Collocare delle funzioni MsgBox in posizioni strategiche, per monitorare il valore di determinate variabili. Fortunatamente, le caselle MsgBox nelle routine Function si aprono quando si esegue la routine. Assicuratevi che solo una formula nel foglio di lavoro utilizzi la vostra funzione. Altrimenti, le caselle MsgBox compariranno per ogni formula valutata e alla lunga questo potrebbe diventare molto fastidioso.
- Sottoporre a test la routine richiamandola da una routine Sub. Gli errori runtime normalmente appaiono in una finestra pop-up e potrete correggere il problema (se sapete qual è) o saltare direttamente nel debugger.
- Impostare un punto di interruzione nella funzione e poi utilizzare il debugger di Excel per controllare la funzione. Avrete così accesso a tutti gli strumenti di debug. Per informazioni sul debugger fate riferimento al Capitolo 13.

Introduzione alle funzioni wrapper

Questo paragrafo contiene alcune funzioni wrapper relativamente semplici che possono essere utili in un'ampia gamma di attività.

Le funzioni wrapper avvolgono, inglobano elementi di codice VBA. In altre parole, consentono di utilizzare le funzioni VBA nelle formule del foglio di lavoro.

Il paragrafo precedente di questo capitolo, "Una funzione senza argomenti", mostra un esempio di funzione wrapper:

```
Function User()
'   Restituisce il nome dell'utente corrente
     User = Application.UserName
End Function
```

Questa funzione consente in pratica di accedere alla proprietà UserName dell'oggetto Application.

Il resto del paragrafo contiene alcune altre funzioni wrapper.

La funzione NumberFormat

Questa funzione mostra semplicemente il formato numerico di una cella. Può essere utile se è necessario assicurarsi che tutte le celle di un gruppo abbiano lo stesso formato numerico.

```
Function NumberFormat(Cell)
'   Restituisce il formato numerico della cella
     NumberFormat = Cell(1).NumberFormat
End Function
```

Notate l'uso di `Cell(1)`? Se come argomento viene utilizzato un intervallo multicella, verrà utilizzata solo la prima cella.

Potete scrivere facilmente funzioni simili che, data una cella, restituiscono il colore del testo, il colore di sfondo, il carattere e così via.

La funzione ExtractElement

Questa funzione wrapper restituisce una sottostringa da una stringa di testo che contiene più elementi, sulla base di un carattere separatore. Per esempio, questa formula restituisce `mucca`, che è il terzo elemento di una stringa che utilizza come separatore lo spazio. Gli argomenti, naturalmente, possono essere riferimenti a celle.

```
=ExtractElement("cane cavallo mucca gatto"; 3; " ")
```

Ecco il codice, che è un wrapper per la funzione VBA Split:

```
Function ExtractElement(Txt, n, Sep)
'   Restituisce l'elemento n-esimo di una stringa, dove gli
'   elementi sono individuati da un carattere separatore
     ExtractElement = Split(Application.Trim(Txt), Sep)(n - 1)
End Function
```

La Figura 20.4 mostra la funzione ExtractElement utilizzata nelle formule del foglio di lavoro. La colonna A contiene la stringa di testo, la colonna B contiene il numero dell'elemento da estrarre e la colonna C contiene il delimitatore (le celle che sembrano vuote contengono un carattere di spazio).

La funzione SayIt

Questa semplice funzione è un wrapper per il metodo Speak dell'oggetto Application.Speech. Usa una voce sintetizzata per "pronunciare" l'argomento.

```
Function SayIt(txt)
     ' Pronuncia l'argomento
     Application.Speech.Speak txt, True
End Function
```

FIGURA 20.4 Uso della funzione ExtractElement per restituire un elemento da una stringa.

La formula controlla il contenuto della cella C10. Se il valore è maggiore di 10.000, la funzione pronuncia il testo: “Supera il budget.” Se il valore è inferiore a 10.000, la funzione visualizza il testo OK (e non dice nulla).

Usatela con parsimonia. Se utilizzate questo genere di funzioni più di una volta, genererete solo confusione. Ricordate inoltre che questa funzione viene valutata a ogni ricalcolo del foglio di lavoro, quindi se state apportando molte modifiche, alla lunga la voce potrebbe diventare molto fastidiosa. Questa funzione è probabilmente più adatta solo per scopi di intrattenimento.

La funzione IsLike

L’operatore VBA Like è un modo molto flessibile per confrontare due stringhe di testo. Verificatene le funzionalità nella Guida di VBA. Questa funzione trasferisce questa potenzialità alle formule del foglio di lavoro:

```
Function IsLike(text, pattern)
'   Restituisce true se il primo argomento è come il secondo
     IsLike = text Like pattern
End Function
```

Funzioni che restituiscono una matrice

Le formule per matrici sono una delle funzionalità più potenti di Excel. Se avete familiarità con le formule per matrici, sarete felici di sapere che potete anche creare funzioni VBA che restituiscono una matrice.

Restituire una matrice dei nomi dei mesi

Un semplice esempio di matrice è la funzione MonthNames. Restituisce una matrice di 12 elementi dei – indovinate… – nomi dei mesi.

```
Function MonthNames()
     MonthNames = Array("Gennaio", "Febbraio", "Marzo", _
       "Aprile", "Maggio", "Giugno", "Luglio", "Agosto", _
       "Settembre", "Ottobre", "Novembre", "Dicembre")
End Function
```

Per utilizzare in un foglio di lavoro la funzione MonthNames, è necessario inserirla come una formula a matrice in 12 celle. Per esempio, selezionate l’intervallo A2: L2 e digitate `=MonthNames()`. Quindi premete Ctrl + Maiusc + Invio per inserire la formula a matrice in tutte le 12 celle selezionate. La Figura 20.5 mostra il risultato.

FIGURA 20.5 La funzione MonthNames restituisce una matrice di 12 elementi.

Se desiderate visualizzare i nomi dei mesi in una colonna, selezionate 12 celle in una colonna e utilizzate questa formula a matrice (non dimenticate di inserirla premendo Ctrl + Maiusc + Invio.)

```
=MATR.TRASPOSTA(MonthNames())
```

Potete anche scegliere un mese dalla matrice. Ecco una formula (non a matrice) che visualizza il quarto elemento della matrice: Aprile.

```
=INDICE(MonthNames(),4)
```

Restituire una lista ordinata

Supponiamo che abbiate un elenco di nomi che volete replicare ordinati in un altro intervallo di celle. Non sarebbe bello avere una funzione del foglio di lavoro che lo faccia per voi?

La funzione personalizzata di questo paragrafo fa proprio questo: prende come argomento un intervallo di celle a colonna singola e restituisce una matrice di celle ordinate. La Figura 20.6 ne mostra il funzionamento. L'intervallo A2:A13 contiene alcuni nomi. L'intervallo C2:C13 contiene questa matrice di formule multicella (ricordate che dovete inserire la formula premendo Ctrl + Maiusc + Invio).

```
=Sorted(A2:A13)
```

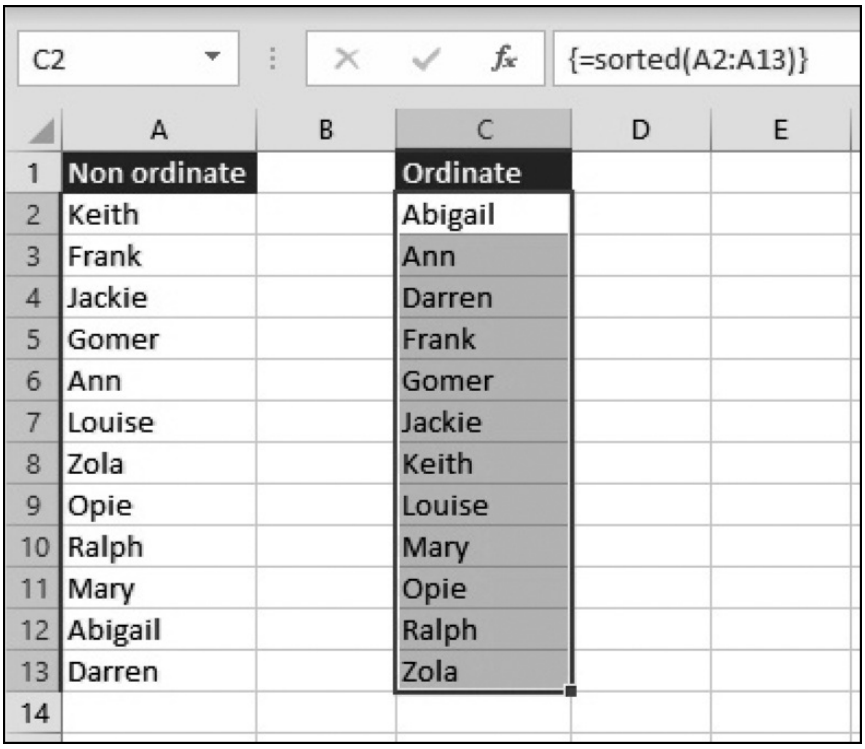

FIGURA 20.6 Uso di una funzione personalizzata per restituire un intervallo ordinato.

Ecco il codice della funzione Sorted:

```
Function Sorted(Rng As Range)
     Dim SortedData() As Variant
     Dim Cell As Range
     Dim Temp As Variant, i As Long, j As Long
     Dim NonEmpty As Long

'   Trasferisce i dati a SortedData
     For Each Cell In Rng
         If Not IsEmpty(Cell) Then
              NonEmpty = NonEmpty + 1
```

```
                ReDim Preserve SortedData(1 To NonEmpty)
                SortedData(NonEmpty) = Cell.Value
            End If
        Next Cell
'   Ordina la matrice
        For i = 1 To NonEmpty
            For j = i + 1 To NonEmpty
                If SortedData(i) > SortedData(j) Then
                  Temp = SortedData(j)
                  SortedData(j) = SortedData(i)
                  SortedData(i) = Temp
                End If
            Next j
        Next i

'   Traspone la matrice e la restituisce
        Sorted = Application.Transpose(SortedData)
End Function
```

La funzione Sorted inizia creando una matrice denominata SortedData. Questa matrice contiene tutti i valori non vuoti dell'intervallo specificato come argomento. Successivamente, la matrice SortedData viene ordinata, utilizzando un algoritmo di tipo *bubble-sort*. Poiché la matrice è di tipo orizzontale, prima di essere restituita dalla funzione deve essere trasposta.

La funzione Sorted opera su un intervallo di celle di qualsiasi dimensione, purché occupi una singola colonna o riga. Se i dati non ordinati sono situati in una riga, la vostra formula deve utilizzare la funzione MATR.TRASPOSTA (TRANSPOSE) di Excel per visualizzare i dati ordinati orizzontalmente. Per esempio:

```
=MATR.TRASPOSTA(Sorted(A16:L16))
```

La finestra di dialogo Inserisci funzione

La finestra di dialogo Inserisci funzione di Excel è uno strumento utile che consente di selezionare una funzione da un elenco e di richiedere gli argomenti necessari. Come ho scritto in precedenza in questo capitolo, le funzioni personalizzate del foglio di lavoro vengono visualizzate anche nella finestra di dialogo Inserisci funzione. Le funzioni personalizzate vengono visualizzate nella categoria Definite dall'utente.

CONSIGLIO

Le routine Function definite con la parola riservata Private non compaiono nella finestra di dialogo Inserisci funzione. Pertanto, se scrivete una routine Function progettata per essere utilizzata solo da altre routine VBA (ma non nelle formule), dichiaratela Private.

Visualizzare la descrizione della funzione

La finestra di dialogo Inserisci funzione visualizza una descrizione per ciascuna funzione standard. Ma come si può vedere nella Figura 20.7, per una funzione personalizzata la descrizione non mostra nulla di utile.

FIGURA 20.7 Per impostazione predefinita, la finestra di dialogo Inserisci funzione non fornisce una descrizione per le funzioni personalizzate.

Per visualizzare una descrizione significativa della funzione personalizzata nella finestra di dialogo Inserisci funzione, dovete eseguire alcuni passaggi aggiuntivi (non esattamente intuitivi).

1. **Attivate un foglio di lavoro nella cartella di lavoro che contiene la funzione personalizzata.**
2. **Selezionate il comando Sviluppo ⇨ Codice ⇨ Macro (o premete Alt + F8).**

   Apparirà la finestra di dialogo Macro.
3. **Nel campo Nome macro, digitate il nome della funzione.**

   Notate che la funzione non appare nell'elenco delle macro; dovete digitarne il nome.
4. **Fate clic sul pulsante Opzioni.**

   Verrà visualizzata la finestra di dialogo Opzioni macro.
5. **Nel campo Descrizione, digitate una descrizione della funzione.**
6. **Fate clic su OK.**
7. **Fate clic su Annulla.**

Ora la finestra di dialogo Inserisci funzione visualizzerà la descrizione della funzione (vedi la Figura 20.8).

FIGURA 20.8 La funzione personalizzata ora visualizza una descrizione.

CONSIGLIO

Per impostazione predefinita, le funzioni personalizzate sono elencate nella categoria Definite dall'utente. Per aggiungere una funzione a una categoria differente, dovete utilizzare VBA. La seguente istruzione aggiunge la funzione TopAvg alla categoria Matematiche e trigonometriche (la categoria 3):

```
Application.MacroOptions Macro:="TopAvg", Category:=3
```

Nella Guida trovate i codici delle altre categorie.

RICORDA!

È necessario eseguire questa istruzione una sola volta. Dopo averla eseguita (e dopo aver salvato la cartella di lavoro), il numero di categoria verrà assegnato alla funzione in modo permanente.

Aggiungere le descrizioni degli argomenti

Quando si usa una funzione standard dalla finestra di dialogo Inserisci funzione, la finestra di dialogo Argomenti funzione visualizza le descrizioni degli argomenti (vedi la Figura 20.9).

Descrizioni degli argomenti delle funzioni

In passato, non era possibile aggiungere le descrizioni degli argomenti. Ma a partire da Excel 2010, Microsoft ha finalmente implementato questa funzionalità.

Fornisci descrizioni degli argomenti utilizzando il metodo MacroOptions. Ecco un esempio che aggiunge le descrizioni per gli argomenti utilizzati dalla funzione TopAvg:

```
Sub AddArgumentDescriptions()
     Application.MacroOptions Macro:="TopAvg", _
         ArgumentDescriptions:= _
         Array("Intervallo che contiene i valori", _
         "Numero dei valori da considerare")
End Sub
```

FIGURA 20.9 Come impostazione predefinita, la finestra di dialogo Argomenti funzione viene visualizzata solo per le funzioni standard.

È necessario eseguire questa routine una sola volta. Dopo averla eseguita, le descrizioni degli argomenti saranno memorizzate nella cartella di lavoro e rimarranno associate alla funzione.

Notate che le descrizioni degli argomenti vengono visualizzate come argomenti della funzione Array. È necessario utilizzare la funzione Array anche se occorre assegnare la descrizione per una funzione con un solo argomento.

Questo capitolo ha fornito molte informazioni sulla creazione di funzioni personalizzate. Utilizzate questi esempi come modelli per creare le vostre funzioni. Come al solito, troverete sulla Guida ulteriori dettagli. Se desiderate scoprire come rendere più accessibili le vostre funzioni personalizzate memorizzandole in un componente aggiuntivo, proseguite la lettura con il Capitolo 21.

Capitolo 21

Creare componenti aggiuntivi di Excel

IN QUESTO CAPITOLO

- **Usare i componenti aggiuntivi**
- **Perché creare nuovi componenti aggiuntivi**
- **Creare componenti aggiuntivi personalizzati**
- **Un esempio di componente aggiuntivo**

Una delle funzionalità più interessanti di Excel è la possibilità di creare dei *componenti aggiuntivi*. I componenti aggiuntivi di Excel consentono di impacchettare le routine VBA e renderle parte dell'interfaccia di Excel. Ma la cosa più bella è che potete condividere questi pacchetti con altri, per fornire loro le vostre routine VBA.

In questo capitolo, vedremo come si creano componenti aggiuntivi utilizzando gli strumenti offerti da Excel.

D'accordo... ma che cos'è un componente aggiuntivo?

Partiamo dall'inizio. Un *componente aggiuntivo* di Excel è qualcosa che si aggiunge per migliorare le funzionalità di Excel. Alcuni componenti aggiuntivi forniscono nuove funzioni per il foglio di lavoro, che potete utilizzare nelle formule; altri forniscono nuovi comandi o strumenti. Se il componente aggiuntivo è progettato correttamente, le nuove funzionalità si integreranno talmente bene con l'interfaccia originale da sembrare elementi del programma.

CONSIGLIO

Excel viene fornito con numerosi componenti aggiuntivi, inclusi Strumenti di analisi e il Risolutore. Inoltre potete ottenere altri componenti aggiuntivi da fornitori terzi o come *shareware*.

Qualsiasi utente esperto è in grado di creare un componente aggiuntivo, ma sono richieste alcune competenze di programmazione VBA. Un componente aggiuntivo di Excel è fondamentalmente una forma particolare di file XLSM. Più in particolare, un componente aggiuntivo è una normale cartella di lavoro XLSM con le seguenti differenze.

- La proprietà IsAddin dell'oggetto Workbook è True.
- La finestra della cartella di lavoro è nascosta e non può essere visualizzata con il comando Visualizza ➪ Finestra ➪ Scopri.
- La cartella di lavoro non è un componente della collezione Workbooks. Al contrario, si trova in AddIns.

Potete convertire qualsiasi file di cartella di lavoro in un componente aggiuntivo, ma non tutte le cartelle di lavoro sono buoni candidati. Poiché i componenti aggiuntivi sono sempre nascosti, non potete visualizzare i fogli di lavoro o i fogli grafici di un componente aggiuntivo. Tuttavia, potete accedere alle routine Sub e Function del componente aggiuntivo e visualizzare le finestre di dialogo contenute in UserForms.

RICORDA!

Per distinguerli dai file XLSM, solitamente i componenti aggiuntivi di Excel hanno l'estensione XLAM. I componenti aggiuntivi delle versioni pre-Excel 2007 hanno l'estensione XLA.

Perché creare dei componenti aggiuntivi?

Potreste decidere di convertire la vostra applicazione Excel in un componente aggiuntivo per uno dei seguenti motivi.

- **Per ostacolare l'accesso al vostro codice.** Quando distribuite un'applicazione sotto forma di componente aggiuntivo (e proteggete il suo progetto VBA), gli utenti occasionali non potranno visualizzare i suoi fogli nella cartella di lavoro. Se nel vostro codice VBA utilizzate tecniche "segrete", potete rendere più difficile la copia di tali parti di codice. Tuttavia le funzionalità di protezione di Excel non sono perfette, e sono disponibili strumenti in grado di aggirare le password.
- **Per evitare la confusione.** Se un utente carica la vostra applicazione come un componente aggiuntivo, il file sarà invisibile e quindi è meno probabile che confonda e disorienti gli utenti inesperti. A differenza di una cartella di lavoro nascosta, il contenuto di un componente aggiuntivo non può essere rivelato.
- **Per semplificare l'accesso alle funzioni del foglio di lavoro.** Le funzioni personalizzate memorizzate in un componente aggiuntivo non richiedono il qualificatore del nome della cartella di lavoro. Per esempio, se memorizzate una funzione personalizzata denominata MOVAVG nella cartella di lavoro NEWFUNC.XLSM, per utilizzare questa funzione in un'altra cartella di lavoro sarebbe necessario utilizzare la sintassi seguente:

  ```
  =NEWFUNC.XLSM!MOVAVG(A1:A50)
  ```

  Ma se questa funzione è memorizzata in un file aperto per un componente aggiuntivo, potrete utilizzare una sintassi molto più semplice, perché non avrete bisogno di specificare il riferimento al file:

  ```
  =MOVAVG(A1:A50)
  ```

- **Per fornire un accesso più semplice per gli utenti.** Dopo aver identificato la posizione del componente aggiuntivo, viene visualizzata nella finestra di dialogo Componenti aggiuntivi, con un nome e una descrizione delle funzionalità da esso svolte. L'utente potrà facilmente abilitare o disabilitare il componente aggiuntivo.
- **Per ottenere un migliore controllo sul caricamento.** I componenti aggiuntivi possono essere aperti automaticamente all'avvio di Excel, indipendentemente dalla directory in cui sono memorizzati.
- **Per evitare di visualizzare dei messaggi durante la chiusura.** Quando un componente aggiuntivo viene chiuso, l'utente non riceve la finestra di dialogo che chiede se salvare le modifiche apportate al file.

Usare i componenti aggiuntivi

Potete caricare e scaricare i componenti aggiuntivi utilizzando la finestra di dialogo Componenti aggiuntivi. Per visualizzare questa finestra di dialogo, selezionate File ➪ Opzioni ➪ Componenti aggiuntivi. Quindi selezionate Componenti aggiuntivi di Excel dall'elenco a discesa nella parte inferiore di questa finestra di dialogo e fate clic su Vai. Oppure prendete la corsia preferenziale e selezionate il comando Sviluppo ➪ Componenti aggiuntivi ➪ Componenti aggiuntivi di Excel. Ma il metodo più semplice consiste semplicemente nel premere Alt + QH.

Uno qualsiasi di questi metodi visualizza la finestra di dialogo Componenti aggiuntivi, illustrata nella Figura 21.1. La casella di riepilogo contiene il nome di tutti i componenti aggiuntivi di cui Excel è a conoscenza. In questo elenco, i segni di spunta identificano eventuali componenti aggiuntivi già aperti. Potete aprire e chiudere i componenti aggiuntivi dalla finestra di dialogo Componenti aggiuntivi, selezionando o deselezionando la loro casella di controllo.

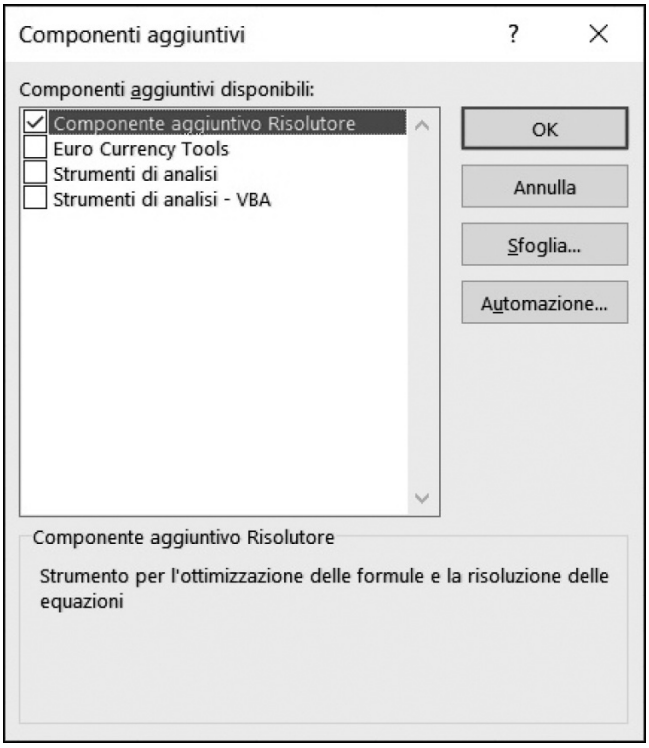

FIGURA 21.1 La finestra di dialogo Componenti aggiuntivi elenca tutti i componenti aggiuntivi noti a Excel.

Per aggiungere un nuovo componente aggiuntivo all'elenco, fate clic su Sfoglia e individuate il file XLAM.

WARNING

Potete anche aprire la maggior parte dei file aggiuntivi (come se fossero file della cartella di lavoro) scegliendo File ➪ Apri ➪ Sfoglia. Un componente aggiuntivo aperto in questo modo non viene visualizzato nella finestra di dialogo Componenti aggiuntivi. Inoltre, se aprite il componente aggiuntivo scegliendo il comando Apri, non potrete chiuderlo scegliendo File ➪ Chiudi. Potete rimuovere il componente aggiuntivo solo uscendo e riavviando Excel o scrivendo una macro apposita per chiudere il componente aggiuntivo.

Quando aprite un componente aggiuntivo, potreste non notare nulla nell'interfaccia utente. In molti casi, tuttavia, la barra multifunzione cambia in qualche modo; Excel visualizza una nuova scheda o uno o più nuovi gruppi in una scheda esistente. Per esempio, l'apertura del componente aggiuntivo Strumenti di analisi fornisce un nuovo elemento nella scheda Dati: Analisi ➪ Analisi dati. Se il componente aggiuntivo contiene solo funzioni personalizzate, le nuove funzioni compariranno nella finestra di dialogo Inserisci funzione, ma nell'interfaccia utente di Excel non noterete alcuna differenza.

Concetti di base sui componenti aggiuntivi

Sebbene sia possibile convertire qualsiasi cartella di lavoro in un componente aggiuntivo, non tutte le cartelle di lavoro beneficeranno di questa conversione. Una cartella di lavoro senza macro rende il componente aggiuntivo inutile. In effetti, gli unici tipi di cartelle di lavoro che traggono vantaggio dall'essere convertiti in componenti aggiuntivi sono quelli con macro. Per esempio, una cartella di lavoro costituita da macro generiche (routine Sub e Function) costituisce un componente aggiuntivo ideale.

Creare un componente aggiuntivo non è difficile, ma richiede ulteriore lavoro. Per creare un componente aggiuntivo da una normale cartella di lavoro, utilizzate la seguente procedura.

1. **Sviluppate la vostra applicazione e assicuratevi che tutto funzioni correttamente.**

   Non dimenticate di includere un metodo per l'esecuzione della o delle macro. Potreste voler definire un tasto di scelta rapida o personalizzare l'interfaccia utente in qualche modo (consultate il Capitolo 19). Se il componente aggiuntivo è costituito solo da funzioni, non è necessario includere un metodo per eseguirle, perché esse compaiono già nella finestra di dialogo Inserisci funzione.

2. **Sottoponete a test l'applicazione eseguendola quando è attiva un'altra cartella di lavoro.**

   In questo modo simulerete il comportamento dell'applicazione quando verrà utilizzata come componente aggiuntivo, poiché un componente aggiuntivo non è mai la cartella di lavoro attiva.

3. **Attivate il VBE e selezionate la cartella di lavoro nel riquadro Progetto; scegliete Strumenti ➪ Proprietà di VBA Project e fate clic sulla scheda Protezione; selezionate la casella di controllo**

Proteggi progetto dalla visualizzazione e specificate (per due volte) la password; quindi fate clic su OK.

Questo passaggio è necessario solo se desiderate impedire ad altri utenti di visualizzare o modificare le vostre macro o UserForm.

4. **Di nuovo in Excel, selezionate il comando File ⇨ Informazioni e, nella parte inferiore del pannello di destra, selezionate Mostra tutte le proprietà.**

   Excel espanderà l'elenco delle proprietà.

5. **Inserite un breve titolo descrittivo nel campo Titolo e una descrizione più estesa nel campo Commenti.**

   I Passi 4 e 5 non sono strettamente necessari, ma semplificano l'utilizzo del componente aggiuntivo, poiché le descrizioni inserite vengono visualizzate nella finestra di dialogo Componenti aggiuntivi quando viene selezionato il componente aggiuntivo.

6. **Sempre in Informazioni, fate clic su Salva con nome nel riquadro di sinistra.**

7. **Fate clic su Sfoglia, nella schermata Salva con nome. Nella finestra di dialogo Salva con nome, selezionate, dall'elenco a discesa Salva come, il tipo Componente aggiuntivo di Excel (*.xlam).**

8. **Specificate la cartella in cui intendete archiviare il componente aggiuntivo.**

   Excel propone la sua cartella predefinita dei componenti aggiuntivi (denominata AddIns), ma potete salvare il file in qualsiasi cartella.

9. **Fate clic su Salva.**

La copia della vostra cartella di lavoro verrà convertita in un componente aggiuntivo e salvata con estensione XLAM. La cartella di lavoro originale rimarrà aperta.

Un esempio di componente aggiuntivo

Questo paragrafo illustra i passaggi necessari per creare un componente aggiuntivo. L'esempio si basa sullo strumento di conversione del testo ChangeCase descritto nel Capitolo 16.

CONSIGLIO

La versione XLSM di questo esempio è disponibile, in originale inglese, sul sito web del libro. Potete creare un componente aggiuntivo partendo da questa cartella di lavoro.

Impostare la cartella di lavoro

La cartella di lavoro è composta da un foglio di lavoro vuoto, un modulo VBA e una UserForm. Il Capitolo 19 spiega come implementare il codice che aggiunge un nuovo elemento al menu contestuale della cella.

La versione originale dello strumento include le opzioni per mettere il testo in lettere maiuscole, minuscole e con le iniziali maiuscole. La versione del componente aggiuntivo include altre due opzioni nella UserForm in modo che abbia le opzioni disponibili in Microsoft Word:

- **Prima parola maiuscola:** solo la prima parola ha l'iniziale maiuscola e tutte le altre lettere sono minuscole.
- **Inversione:** tutte le lettere maiuscole vengono convertite in lettere minuscole e viceversa.

La Figura 21.2 mostra la UserForm1. I cinque controlli OptionButton si trovano all'interno di un controllo Frame. Inoltre, la UserForm1 ha un pulsante Annulla (denominato CancelButton) e un pulsante OK (denominato OKButton).

FIGURA 21.2 La UserForm per il componente aggiuntivo ChangeCase.

Il codice eseguito quando fate clic sul pulsante Annulla è molto semplice. Questa routine chiude la UserForm senza nessuna azione:

```
Private Sub CancelButton_Click()
     Unload UserForm1
End Sub
```

Ecco invece il codice che viene eseguito quando fate clic sul pulsante OK. Questo codice esegue tutto il lavoro:

```
Private Sub OKButton_Click()
     Dim TextCells As Range
     Dim cell As Range
     Dim Text As String
     Dim i As Long

'   Crea un oggetto con le sole costanti testuali
     On Error Resume Next
     Set TextCells = Selection.SpecialCells(xlConstants, xlTextValues)

'   Disattiva l’aggiornamento dello schermo
     Application.ScreenUpdating = False

'   Ciclo sulle celle
     For Each cell In TextCells
         Text = cell.Value
         Select Case True
         Case OptionLower 'minuscole
              cell.Value = LCase(cell.Value)
         Case OptionUpper 'MAIUSCOLE
              cell.Value = UCase(cell.Value)
         Case OptionProper 'Iniziali Maiuscole
              cell.Value = WorksheetFunction.Proper(cell.Value)
         Case OptionSentence 'Prima parola maiuscola
              Text = UCase(Left(cell.Value, 1))
              Text = Text & LCase(Mid(cell.Value, 2, Len(cell.Value)))
              cell.Value = Text
         Case OptionToggle 'iNVERSIONE
              For i = 1 To Len(Text)
                If Mid(Text, i, 1) Like "[A-Z]" Then
                  Mid(Text, i, 1) = LCase(Mid(Text, i, 1))
                Else
                  Mid(Text, i, 1) = UCase(Mid(Text, i, 1))
                End If
              Next i
              cell.Value = Text
         End Select
     Next

'   Chiude la finestra di dialogo
     Unload UserForm1
End Sub
```

Oltre alle due nuove opzioni, questa versione dello strumento Maiuscole/minuscole differisce dalla versione nel Capitolo 16 in altri due modi.

- Il metodo SpecialCells crea una variabile oggetto composta dalle sole celle della selezione che contengono una costante testuale (e non una formula). Questa tecnica rende la routine più veloce se la selezione contiene molte celle con formule. Per ulteriori informazioni su questa tecnica consultate il Capitolo 14.
- A causa dell'opzione Maiuscole/minuscole dei menu contestuali delle righe e delle colonne, è possibile eseguire lo strumento facendo clic destro su un intervallo, sulla selezione di un'intera riga o sulla selezione di un'intera colonna. Ciascuna di queste azioni visualizza nel menu contestuale la voce Maiuscole/minuscole.

Provare la cartella di lavoro

Sottoponete a test il componente aggiuntivo prima di convertire questa cartella di lavoro. Per simulare ciò che accade quando la cartella di lavoro è un componente aggiuntivo, è necessario provare la cartella di lavoro quando è attiva un'altra cartella di lavoro. Poiché un componente aggiuntivo non è mai il foglio o la cartella di lavoro attivo, il fatto di provarlo quando è aperta un'altra cartella di lavoro può aiutare a identificare alcuni potenziali errori.

1. **Aprite una nuova cartella di lavoro e inserite informazioni varie in alcune celle.**

   Come test, inserite vari tipi di informazioni, fra cui testi, valori e formule. Oppure aprite semplicemente una cartella di lavoro esistente e usatela come strumento di test. Ricordatevi che qualsiasi modifica apportata alla cartella di lavoro non potrà essere annullata, quindi è sempre meglio usare una copia.

2. **Selezionate una o più celle (o intere righe e colonne).**
3. **Eseguite la macro ChangeCase scegliendo il nuovo comando Maiuscole/minuscole dal menu contestuale della cella (o della riga o della colonna).**

RICORDA!

Se nel menu contestuale non compare il comando Maiuscole/minuscole, la causa più probabile è che non abbiate abilitato le macro all'apertura della cartella di lavoro. Chiudete la cartella di lavoro e poi riapritela, assicurandovi di abilitare le macro.

Aggiungere informazioni descrittive

Sebbene non sia obbligatorio, è quanto meno corretto inserire una descrizione del componente aggiuntivo. Per aggiungere una descrizione seguite questi passaggi.

1. **Attivate la cartella di lavoro change case.xlsm.**
2. **Selezionate il comando File ➪ Informazioni e fate clic su Mostra tutte le proprietà, in basso a destra.**

   Excel espanderà l'elenco delle proprietà.

3. **Inserite nel campo Titolo un titolo per il componente aggiuntivo.**

   Questo testo viene visualizzato nell'elenco di componenti, nella finestra di dialogo Componenti aggiuntivi. Per questo esempio, inserite Maiuscole/minuscole.

4. **Nel campo Commenti, inserite una descrizione.**

   Queste informazioni vengono visualizzate nella parte inferiore della finestra di dialogo Componenti aggiuntivi quando viene selezionato il componente. Per questo esempio, immettere **Cambia le lettere maiuscole/minuscole delle parole**. Potete accedere a questo strumento dal menu contestuale.

La Figura 21.3 mostra la sezione Proprietà con i campi Titolo e Commenti compilati.

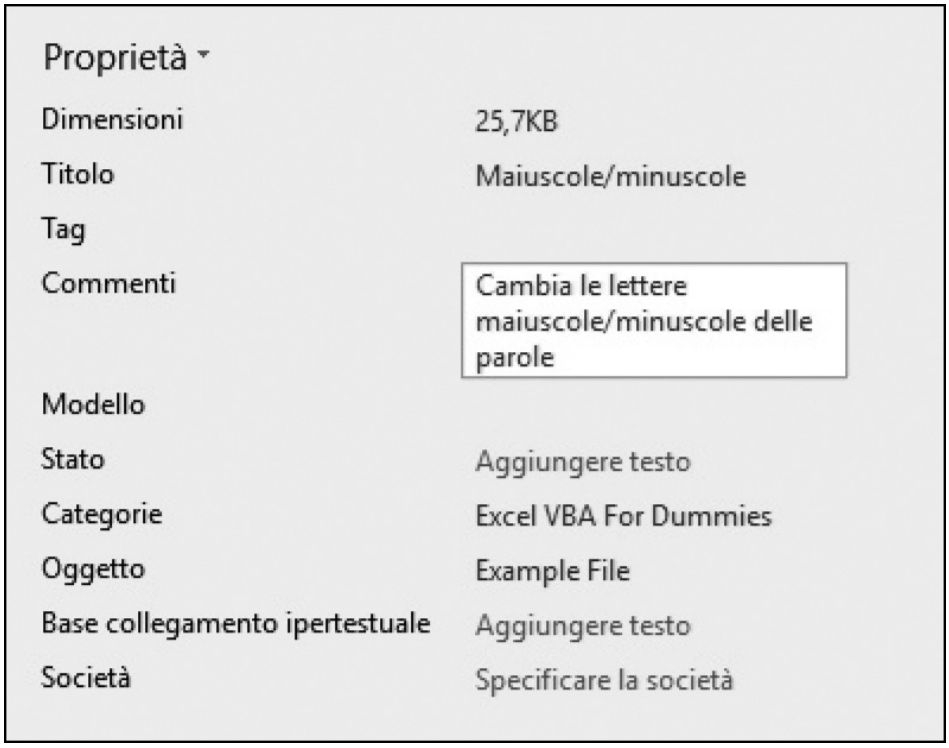

FIGURA 21.3 Utilizzate la sezione Proprietà per immettere informazioni descrittive sul componente aggiuntivo.

Proteggere il codice VBA

Se volete aggiungere una password per impedire ad altri di visualizzare il vostro codice VBA, utilizzate la seguente procedura.

1. **Attivate il VBE e selezionate la cartella di lavoro change case.xlsm nel riquadro Progetto.**
2. **Scegliete Strumenti ➪ Proprietà di VBAProject e, nella finestra di dialogo che apparirà, fate clic sulla scheda Protezione.**
3. **Selezionate la casella di controllo Proteggi progetto dalla visualizzazione e digitate (per due volte) una password.**
4. **Fate clic OK.**
5. **Salvate la cartella di lavoro con File ➪ Salva dal VBE o dalla finestra di Excel.**

Creare il componente aggiuntivo

A questo punto, avete sottoposto a test il file change case.xlsm, e siete certi che funzioni correttamente. Il passo successivo è la creazione del componente aggiuntivo. Ecco come dovete procedere.

1. **Se necessario, riattivate Excel.**
2. **Attivate la cartella di lavoro change case.xlsm e selezionate il comando File ➪ Salva con nome ➪ Sfoglia.**

   Excel visualizzerà la finestra di dialogo Salva con nome.
3. **Dal menu a discesa Salva come, selezionate Componente aggiuntivo.**
4. **Specificate il percorso e fate clic su Salva.**

   Verrà creato un nuovo file aggiuntivo (con estensione `.xlam`) ma rimarrà aperta la versione originale XLSM.

Aprire il componente aggiuntivo

Per evitare confusione, chiudete la cartella di lavoro XLSM prima di aprire il componente aggiuntivo che avete creato a partire da tale cartella di lavoro.

Aprite il componente aggiuntivo nel seguente modo.

1. **Selezionate il comando Sviluppo ➪ Componenti aggiuntivi ➪ Componenti aggiuntivi di Excel (o premete Alt + QH).**

   Excel visualizzerà la finestra di dialogo Componenti aggiuntivi.
2. **Fate clic sul pulsante Sfoglia.**

3. **Individuate e selezionate il componente aggiuntivo che avete appena creato.**
4. **Fate clic su OK per chiudere la finestra di dialogo Sfoglia.**

   Dopo aver trovato il nuovo componente aggiuntivo, la finestra di dialogo Componenti aggiuntivi elencherà il componente aggiuntivo. Come si vede nella Figura 21.4, la finestra di dialogo Componenti aggiuntivi mostra anche le informazioni descrittive specificate.

5. **Assicuratevi che, nella finestra di dialogo Componenti aggiuntivi, il vostro nuovo componente aggiuntivo sia selezionato.**
6. **Fate clic su OK per chiudere la finestra di dialogo.**

   Excel aprirà il componente aggiuntivo. Ora potete usarlo in tutte le vostre cartelle di lavoro. Finché rimane selezionato nella finestra di dialogo Componenti aggiuntivi, ogni volta che avvierete Excel aprirete anche il componente aggiuntivo.

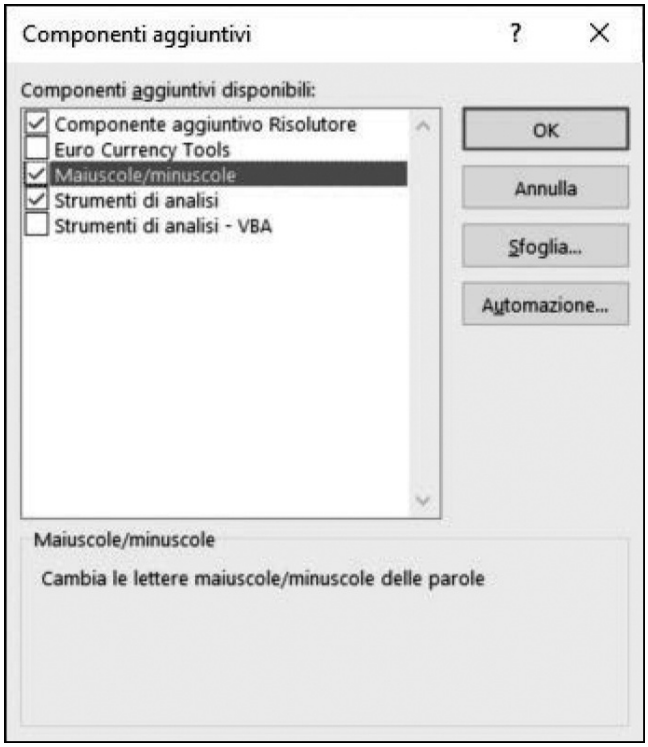

FIGURA 21.4 La finestra di dialogo Componenti aggiuntivi ha il nuovo componente aggiuntivo selezionato.

Distribuire il componente aggiuntivo

Se siete un tipo generoso, potete anche pensare di distribuire questo componente aggiuntivo ad altri utenti, fornendo loro una copia del file XLAM (non hanno bisogno della versione XLSM). Quando apriranno il componente aggiuntivo e selezioneranno un intervallo, una o più righe o una o più colonne, nel loro menu contestuale comparirà il vostro nuovo comando Maiuscole/minuscole. Se poi proteggerete il progetto VBA con una password, nessun altro potrà vedere il codice della macro, a meno che non conosca la password.

Modificare il componente aggiuntivo

Un componente aggiuntivo può essere modificato come qualsiasi altra cartella di lavoro. Potete modificare direttamente il file XLAM (non è necessario lavorare con la versione XLSM originale) seguendo questi passaggi.

1. **Aprite il vostro file XLAM, se non è già aperto.**
2. **Attivate il VBE.**
3. **Fate doppio clic sul nome del progetto nel riquadro Progetto.**

   Se avete protetto il codice, vi verrà chiesta la password.

4. **Digitate la password e fate clic su OK.**
5. **Apportate al codice le modifiche desiderate.**
6. **Salvate il file scegliendo File ➪ Salva.**

CONSIGLIO

Se create un componente aggiuntivo che memorizza informazioni in un foglio di lavoro, dovrete impostare la proprietà IsAddIn della cartella di lavoro su False per far comparire la cartella di lavoro nella finestra Proprietà quando è selezionato l'oggetto Questa_cartella_di_lavoro (vedi la Figura 21.5). Dopo aver apportato le modifiche alla cartella di lavoro, prima di salvare il file, assicuratevi di impostare nuovamente la proprietà IsAddIn su True.

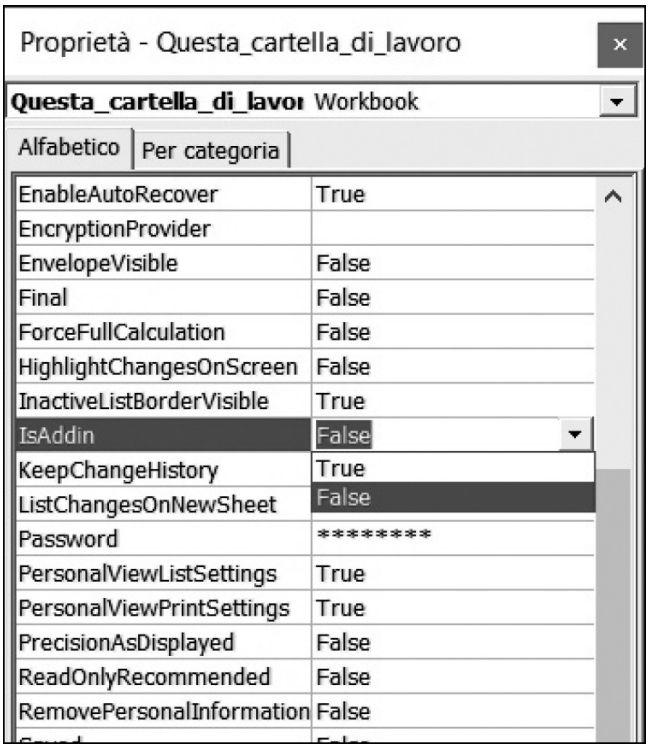

FIGURA 21.5 Togliete temporaneamente al componente la qualifica di "componente aggiuntivo".

6

La parte dei dieci

IN QUESTA PARTE...

- Suggerimenti per l'utilizzo del VBE.
- Ampliare le conoscenze sulle risorse di Excel.
- Che cosa dovreste e non dovreste fare in Excel VBA.

Capitolo 22

Dieci suggerimenti per l'editor di Visual Basic

IN QUESTO CAPITOLO

- **Applicare commenti a blocco**
- **Copiare più righe di codice contemporaneamente**
- **Saltare tra moduli e routine**
- **Teletrasportare le funzioni**
- **Rimanere nella routine corretta**
- **Attraversare il codice**
- **Passare a una riga ben precisa del codice**
- **Fermare il codice in un punto**
- **Vedere per intero il valore delle variabili**
- **Disattivare il controllo automatico della sintassi**

Se userete a lungo il Visual Basic Editor, ecco alcuni strumenti integrati che possono semplificare il vostro lavoro. Che siate analisti, neofiti della programmazione o stanchi veterani che lavorano in un eremo fra le montagne, questi suggerimenti potranno migliorare anche notevolmente la vostra esperienza nella programmazione di macro.

Inserire blocchi di commenti

Posizionando un apice davanti a una qualsiasi istruzione, chiedete a Excel di saltare quella frase: avete trasformato un "pezzo" di codice in un commento. Ma potete utilizzare l'apice anche per scrivere i vostri commenti o delle note al codice (vedi la Figura 22.1).

FIGURA 22.1 Un apice davanti a una qualsiasi riga trasforma quella riga in un commento.

A volte è utile trasformare in commenti più righe di codice. In questo modo, potete provare determinate righe di codice chiedendo a Excel di ignorare le righe di commento.

Invece di perdere tempo a trasformare in commenti le righe una per volta, potete utilizzare la barra degli strumenti Modifica per trasformare in commenti anche interi blocchi di codice.

Potete attivare la barra degli strumenti Modifica, selezionando il comando Visualizza ➪ Barre degli strumenti ➪ Modifica.

Selezionate le righe di codice che desiderate trasformare in commenti e poi fate clic sull'icona Commento, sulla barra degli strumenti Modifica (vedi la Figura 22.2).

CONSIGLIO

Potete assicurarvi che la barra degli strumenti Modifica sia sempre visibile trascinandola sulla barra dei menu del VBE. Si fisserà nella posizione scelta.

Copiare più righe di codice contemporaneamente

Potete copiare interi blocchi di codice evidenziando le righe che vi servono, poi tenendo premuto il tasto Ctrl e infine trascinando il blocco dove vi serve. Questo è un vecchio trucco di Windows che funziona anche nei trascinamenti da un modulo all'altro.

Potete capire che state trascinando una copia perché accanto al puntatore del mouse compare il simbolo "+", come mostrato nella Figura 22.3.

FIGURA 22.2 Con la barra degli strumenti Modifica potete selezionare interi blocchi di codice e di applicare i commenti a tutte le righe, contemporaneamente.

FIGURA 22.3 Tenendo premuto il tasto Ctrl mentre trascinate del codice, ne create una copia.

Spostarsi tra moduli e routine

Mano a mano che la massa del codice che compone la macro inizia ad aumentare, può diventare una vera pena spostarsi rapidamente tra i moduli e le routine. Potete alleviare il dolore usando alcune scorciatoie:

- Premete Ctrl + Tab per spostarvi rapidamente tra i moduli.
- Premete Ctrl + Pag↑ e Ctrl + Pag↓ per spostarvi tra le routine all'interno di un modulo.

Teletrasportare le funzioni

Quando si rivede una macro, a volte si incontra il nome di una variabile o di una funzione che punta a un'altra porzione di codice. Invece di scorrere tutti i moduli per scoprire da dove proviene quella funzione o variabile, potete semplicemente posizionare il cursore sul suo nome e premere Maiusc + F2.

Come vedete nella Figura 22.4, verrete immediatamente teletrasportati all'origine di quella funzione o variabile.

```
nd check for version match
ror GoTo ServerFail
("ReferenceTables").Visible = True

= "SELECT Version FROM RefVersionCheck"
rdset.Open MySql, MyConnection, adOpenStatic, adLock

st MyConnection As String = "Provider=sqloledb;" & _
 "Data Source=prdss2kic004,1433;" & _
 "database=sops;" & _
 "User ID=sopsuser;" & _
```

FIGURA 22.4 Premendo Maiusc + F2 con il cursore su un nome di funzione o di variabile, verrete reindirizzati al punto in cui è definita.

Premendo Ctrl + Maiusc + F2 verrete poi ricondotti al punto iniziale.

Rimanere nella routine corrente

Quando i moduli contengono più routine, può risultare difficile scorrere una determinata routine senza finire inavvertitamente in un'altra routine. Spesso vi ritrovate a scorrere il listato verso l'alto, poi verso il basso, cercando disperatamente di tornare al codice sul quale stavate lavorando.

Per evitare questo lavorio inutile, potete fare clic sul pulsante Visualizza routine nell'angolo in basso a sinistra del VBE (vedi la Figura 22.5). Facendo clic su questo pulsante limitate ogni scorrimento alla sola routine in cui vi trovate.

FIGURA 22.5 Fate clic sul pulsante Visualizza routine per limitare lo scorrimento alla sola routine attiva.

Esecuzione passo-passo del codice

VBA offre diversi strumenti per aiutarvi nel debugging del codice. In programmazione, eseguire il *debugging* significa trovare e correggere gli errori presenti nel codice.

Uno degli strumenti di debug più utili è la possibilità di eseguire il codice una riga alla volta. In questo modo potrete seguire l'esecuzione di ogni singola riga, in isolamento.

Per eseguire passo passo il codice, posizionate il cursore in un punto qualsiasi della macro e premete F8. La macro entrerà in modalità di debug.

Verrà evidenziata la prima riga di codice "attivo" e sul margine sinistro della finestra comparirà una piccola freccia (vedi la Figura 22.6). Premete nuovamente F8 ed eseguirete la riga di codice evidenziata, per poi saltare alla riga successiva. Potete continuare a premete F8 per eseguire il codice, riga dopo riga, fino alla fine della macro.

Inoltre, mentre eseguite passo-passo il codice, potete posizionare il mouse su una qualsiasi stringa o variabile intera per scoprire il valore che assume al momento quella variabile.

Per uscire dalla modalità di debug, scegliete Debug ➪ Esci da istruzione/routine.

CONSIGLIO

Per informazioni dettagliate sul debugging del codice VBA, consultate il Capitolo 13.

FIGURA 22.6 Premete il tasto F8 per eseguire il codice riga per riga.

Passare a una determinata riga del codice

Come abbiamo appena visto, potete eseguire passo passo il codice posizionando il cursore in un punto qualsiasi della macro e premendo F8.

Questo è grandioso, ma che cosa succede se desiderate iniziare a controllare il codice da una riga ben precisa? Potete farlo, spostando semplicemente la freccia!

Quando una riga di codice è evidenziata in modalità di debug, potete fare clic e trascinare la freccia posta sul margine sinistro della finestra Codice verso l'alto o verso il basso, rilasciandola poi in corrispondenza della riga di codice che desiderate eseguire (vedi la Figura 22.7.)

```
'   Esegue la routine per ogni riga di MyCmd
    For Each MyCell In MyRange
        MyCmd.Parameters("@Acctid") = MyAccount
        MyCmd.Parameters("@FType") = MyCell.Offse
        MyCmd.Parameters("@SKey") = MyCell.Offset
        MyCmd.Parameters("@Pd4") = MyCell.Offset
        MyCmd.Parameters("@Pd5") = MyCell.Offset
        MyCmd.Parameters("@Pd6") = MyCell.Offset
```

FIGURA 22.7 Potete fare clic e trascinare la freccia gialla mentre eseguite passo passo il codice.

Arrestare il codice in un punto predefinito

Un altro utile strumento di debug è la possibilità di impostare un punto di interruzione nel codice. Quando si imposta un punto di interruzione, il codice viene eseguito normalmente, ma poi si arresta alla riga di codice in cui avete definito il punto di interruzione.

Questa tecnica di debug è utile quando si vogliono eseguire test su piccoli blocchi di codice alla volta. Se, per esempio, sospettate dell'esistenza di un errore nella macro ma siete certi che la maggior parte della macro venga eseguita senza problemi, potete impostare un punto di interruzione a partire dalla riga di codice sospetta e poi eseguire la macro. Quando la macro raggiungerà il punto di interruzione, l'esecuzione si ferma. A questo punto, potrete premete F8 per controllare l'esecuzione della macro una riga alla volta.

Per impostare un punto di interruzione nel codice, posizionate il cursore nel punto desiderato e premete F9. Come mostra la Figura 22.8, il VBE segna chiaramente il punto di interruzione con un punto nel margine sinistro della finestra Codice, e la riga di codice stessa viene evidenziata in rosso scuro.

FIGURA 22.8 Un punto di interruzione è contrassegnato da un punto sul margine sinistro e da un'ombreggiatura.

RICORDA!

Quando la macro raggiunge un punto di interruzione, entra davvero in modalità di debug. Per uscire dalla modalità di debug, scegliete Debug ➪ Esci da istruzione/routine.

Vedere per intero il valore delle variabili

Nel VBE, se in modalità di debug passate con il mouse su una stringa o su una variabile di tipo intero, potete visualizzarne il valore in un piccolo riquadro. Diventa così possibile visualizzare i valori che vengono passati e restituiti alle variabili: questo piccolo aiuto può essere molto utile.

Tuttavia, questi “suggerimenti” possono contenere solo 77 caratteri (incluso il nome della variabile). Questo significa che se il valore della variabile è troppo lungo, viene troncato.

Per visualizzare gli ultimi 77 caratteri, tenete premuto il tasto Ctrl mentre vi spostate con il mouse.

La Figura 22.9 mostra l’aspetto di questa funzionalità passando con il mouse su una variabile in modalità di debug.

FIGURA 22.9 Mostrare tutto il valore di una variabile.

Disattivare il controllo automatico della sintassi

Spesso, mentre si lavora su un frammento di codice, si scopre che è necessario copiare qualcosa da un’altra riga. Ma non avete ancora completato la riga, dovete solo lasciarla per un secondo. Ma il VBE vi ferma immediatamente con un messaggio d’errore, simile a quello rappresentato nella Figura 22.10, che sicuramente conoscete.

Questo messaggio vi obbliga a interrompere ciò che state facendo per riconoscere l’errore facendo clic su OK. Dopo mezza giornata di queste brusche interruzioni… diciamo che non ne potrete più.

```
If Application.CountA(Columns(iCounter).
EntireColumn) = 0 Then
    Columns(iCounter).Delete
End If
```

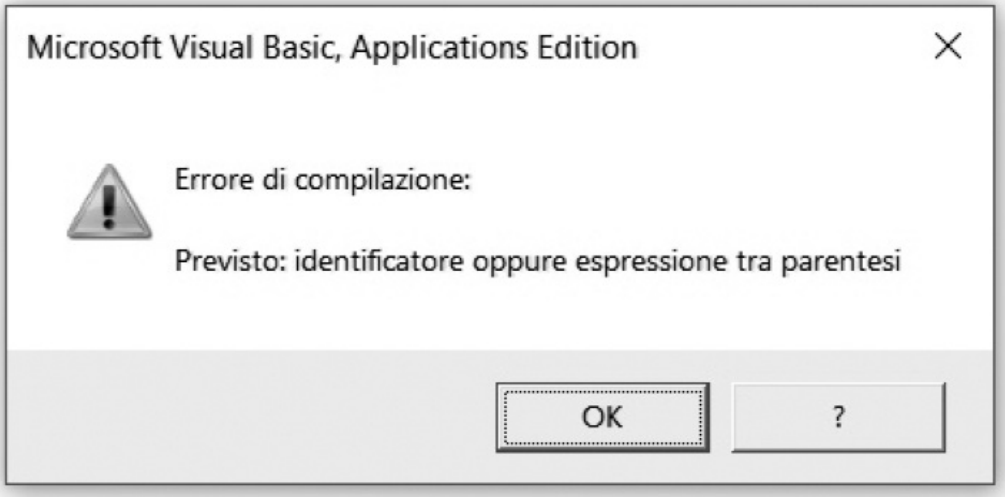

FIGURA 22.10 Lasciare una riga di codice incompleta, anche per un secondo, genera un messaggio d'errore sgradevole.

Bene, potete salvare il vostro equilibrio mentale disattivando l'opzione Controllo automatico sintassi, selezionando il comando Strumenti ➪ Opzioni.

Nella scheda Editor della finestra di dialogo Opzioni, deselezionate l'opzione Controllo automatico sintassi (Figura 22.11) e interromperete questi fastidiosi messaggi di errore.

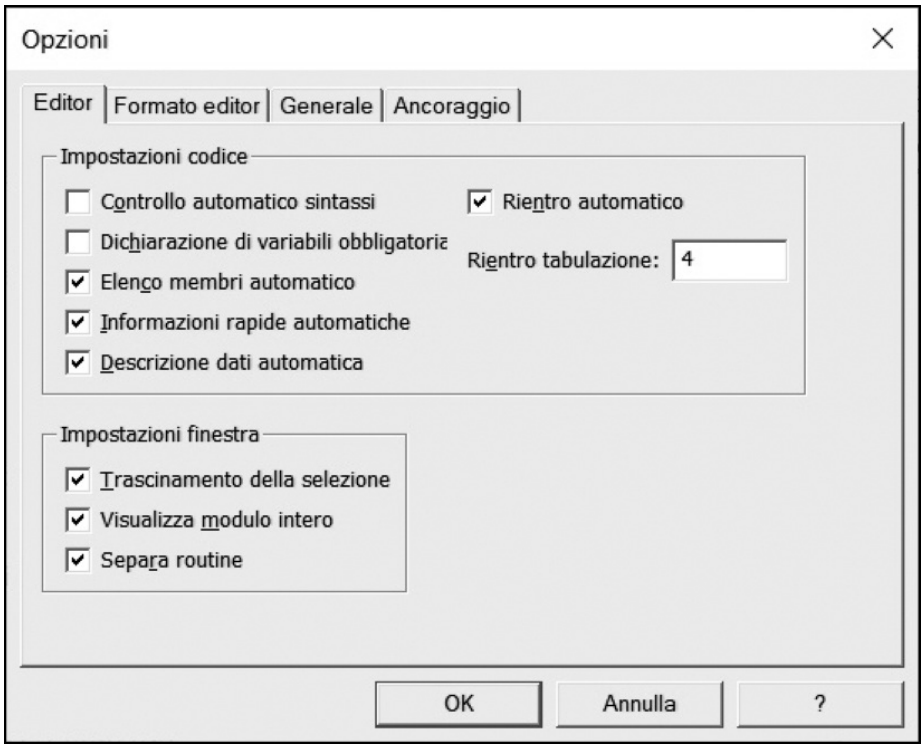

FIGURA 22.11 Deselezionate l'opzione Controllo automatico sintassi per evitare questi messaggi in fase di programmazione.

Non dovete preoccuparvi troppo di perdervi qualche errore. In caso di errore il codice diventa rosso, fornendovi un'indicazione visiva che qualcosa non va.

Capitolo 23

La Guida di VBA

IN QUESTO CAPITOLO

- **Lasciare che Excel scriva il codice per voi**
- **Utilizzare la Guida di VBA**
- **Trarre codice da Internet**
- **Sfruttare i forum di utenti**
- **Visitare i blog degli esperti**
- **Video su YouTube**
- **Frequentare corsi di formazione dal vivo e online**
- **Imparare dal Microsoft Office Dev Center**
- **Sezionare i file Excel disponibili in azienda**
- **Chiedere al vostro guru di Excel**

Non si diventa esperti di VBA in un giorno. Il linguaggio VBA richiede tempo e pratica. La buona notizia è che sono molte le risorse che possono aiutarvi nel percorso. La cattiva è che molte sono in inglese. In questo capitolo, scoprirete dieci dei luoghi più utili cui potete rivolgervi quando avete bisogno di una "dritta".

Lasciare che Excel scriva il codice per voi

Uno dei modi migliori per ottenere un aiuto è il registratore di macro di Excel. Quando preparate una macro con il registratore di macro, Excel scrive il codice VBA per voi. Dopo la registrazione, potrete rivedere il codice, vedere che cosa ha scritto il registratore e poi provare a trasformare il codice in modo da creare qualcosa di più adatto alle vostre esigenze.

Supponiamo, per esempio, che abbiate bisogno di una macro che aggiorni tutte le tabelle pivot della cartella di lavoro e cancelli tutti i filtri in ogni tabella pivot. Scrivere questa macro da zero sarebbe un compito scoraggiante. Ma potete avviare il registratore di macro e registrarvi mentre aggiornate tutte le tabelle pivot e cancellate tutti i filtri. Dopo aver fermato la registrazione, potrete rivedere la macro e apportarle tutte le modifiche che ritenete necessarie.

CONSIGLIO

Avete bisogno di rinfrescare la memoria sull'utilizzo del registratore di macro? Tornate al Capitolo 6, dove troverete una panoramica dettagliata.

Per un nuovo utente di Excel, il sistema della Guida può sembrare un componente aggiuntivo goffo che restituisce un elenco di argomenti che non ha nulla a che fare con l'argomento originale cercato. La verità, tuttavia, è che dopo aver imparato a utilizzare efficacemente la Guida Excel, spesso essa rappresenta il modo più semplice e veloce per ottenere informazioni su un argomento.

Ci sono due principi base nell'uso della Guida Excel.

- **Quando si accede alla Guida, la posizione è importante.** Excel ha due sistemi di documentazione: una Guida che fornisce assistenza sulle funzionalità di Excel e un'altra che fornisce assistenza su argomenti di programmazione VBA. Invece di avviare una ricerca globale in base ai vostri criteri, Excel la avvia nella Guida rilevante, in base alla vostra posizione attuale. Ciò significa, essenzialmente, che l'aiuto che ottenete dipende dall'area di Excel in cui state lavorando. Quindi, se avete bisogno di aiuto su un argomento che

riguarda le macro e la programmazione VBA, occorre essere in VBE prima di lanciare la ricerca. Ciò garantisce che essa venga eseguita nella Guida corretta.

» **La Guida online è migliore della Guida offline.** Quando cercate aiuto su un argomento, Excel controlla la connessione a Internet. Se siete online, Excel restituisce i risultati della Guida in base ai contenuti online tratti dal sito web di Microsoft. In caso contrario, Excel utilizza i file della Guida archiviati localmente con Microsoft Office. La Guida online è generalmente migliore perché il contenuto è più dettagliato e include informazioni più aggiornate, nonché collegamenti ad altre risorse che non sono ovviamente disponibili offline.

Trarre codice da Internet

Il vero segreto nell'era di Internet nell'ambito della programmazione è che non esiste più del "codice originale". Tutta la sintassi per macro di cui avete bisogno è già stata documentata da qualche parte in Internet. L'evoluzione della programmazione richiede meno scrittura del codice "da zero" e più impiego di codice preesistente, da applicare in modo creativo alla situazione.

Se vi sentite bloccati e cercate una macro per svolgere una particolare attività, sottoponete l'attività che state cercando di realizzare al vostro motore di ricerca preferito. Per risultati migliori, prima della descrizione inserite le parole "Excel VBA".

Se, per esempio, state provando a scrivere una macro che cancella tutte le righe vuote di un foglio di lavoro, cercate "Excel VBA cancellare righe vuote foglio lavoro" (o "Excel VBA delete blank rows in a worksheet" se masticate l'inglese). Potete scommettere due mesi di stipendio che qualcuno ha già avuto lo stesso problema e ha pubblicato il codice e le informazioni necessarie per creare una macro per eseguire l'operazione.

Sfruttare i forum degli utenti

Se vi trovate in un vicolo cieco, ponete una domanda in un forum per ottenere indicazioni personalizzate in base alle vostre esigenze.

I forum sono community online che ruotano attorno a un argomento specifico. In questi forum potete porre domande e disporre di esperti che danno consigli su come risolvere particolari problemi. Le persone che rispondono alle domande sono in genere volontari appassionati, che aiutano gli altri a risolvere le sfide poste dal mondo reale.

Sono molti i forum dedicati alle tematiche di Excel. Per trovare un forum su Excel, chiedete aiuto al vostro motore di ricerca.

Ecco alcuni suggerimenti per ottenere il massimo dai forum.

» **Prima di iniziare, leggete sempre (e poi seguite) le regole del forum.** Queste regole spesso includono consigli sulla pubblicazione di domande e linee guida sull'etichetta della community.

» **Controllate se la vostra domanda è stata già posta e se ha già avuto una risposta.** Perché non risparmiare tempo sfruttando l'archivio di domande del forum? Dedicate qualche tempo a cercare nel forum se la vostra domanda è già stata posta da qualcun altro.

» **Usate titoli concisi e precisi nelle vostre domande.** Non ponete domande sul forum con titoli troppo generici, come: "Ho bisogno di un consiglio" o "Aiuto per favore".

» **Tenete ben concentrato il raggio delle vostre domande.** Non fate domande del tipo: "Come faccio a costruire una macro di fatturazione per Excel".

» **Siate pazienti.** Ricordate che chi risponde alle vostre domande è un volontario, che in genere ha un suo lavoro. Date alla comunità il tempo di rispondere alla vostra domanda.

» **Controllate spesso.** Dopo aver inviato la domanda, potreste ricevere richieste di dettagli sulle vostre esigenze. Fate un favore a tutti e tornate spesso sul messaggio per esaminare le risposte o rispondere alle domande di precisazione.

» **Ringraziate chi ha risposto alla vostra domanda.** Se ricevete una risposta, dedicate un momento a un ringraziamento a chi è stato così gentile da aiutarvi.

Visitare blog di esperti

Ci sono alcuni guru di Excel che condividono all'interno dei loro blog le loro conoscenze. Questi blog sono spesso ricchi di suggerimenti e trucchi, che possono aiutarvi a migliorare le vostre abilità e oltretutto sono gratuiti!

Sebbene questi blog non descrivano necessariamente quelle che possono essere le vostre particolari esigenze, offrono articoli che promuovono la conoscenza di Excel e possono dare indicazioni generali sull'impiego di applicazioni per Excel in situazioni aziendali concrete.

Ecco un elenco di alcuni dei migliori blog, a oggi, dedicati a Excel.

» **ExcelGuru:** Ken Puls è un MVP di Microsoft Excel che condivide le conoscenze sul suo blog

http://www.excelguru.ca/blog. Oltre al suo blog, Ken offre diverse risorse di apprendimento, per far progredire le vostre conoscenze in Excel.

- **Chandoo.org:** Purna "Chandoo" Duggirala è un MVP Microsoft Excel indiano, che è entrato in scena nel 2007. Il suo blog innovativo (http://chandoo.org/) offre molti modelli e articoli gratuiti, finalizzati a "rendervi fantastici, con Excel".
- **Contextures:** Debra Dalgleish è un MVP di Microsft Excel e titolare di un noto sito su Excel, http://www.contextures.com. Con un elenco alfabetico di oltre 350 argomenti su Excel, siete sicuri di trovare qualcosa di vostro interesse.
- **DailyDose:** Dick Kusleika è titolare del blog più longevo su Excel: http://www.dailydoseofexcel.com. È il re dei blog su VBA per Excel, con oltre dieci anni di articoli ed esempi.
- **MrExcel:** Bill Jelen è grande ambasciatore di Excel. Questo MVP di Excel di lunga data offre sul suo sito oltre un migliaio di video gratuiti e un'enorme libreria di risorse dedicate alla formazione: http://www.mrexcel.com.

Video su YouTube

Se pensate di preferire la formazione video rispetto agli articoli, considerate l'opportunità di andare su YouTube. Troverete decine di canali gestiti da persone straordinarie che hanno in comune una passione per la condivisione delle conoscenze. Rimarrete sorpresi dalla quantità di tutorial video gratuiti e di alta qualità presenti. Andate su www.youtube.com e cercate "Excel VBA".

Partecipare in diretta e online ai corsi di formazione

I corsi di formazione dal vivo e online sono un modo fantastico per assorbire conoscenze su Excel da un gruppo eterogeneo di persone. Non solo il docente vi insegnerà le tecniche, ma le vivaci discussioni che si scateneranno durante le lezioni sono sempre una ricca fonte di idee e di consigli che non avreste mai nemmeno immaginato. Se vi piace l'energia degli eventi dal vivo, considerate la possibilità di cercare corsi su Excel.

Ecco alcuni siti che offrono eccellenti corsi su Excel:

- http://academy.excelhero.com/excel-hero-academy-tuition
- http://chandoo.org/wp/vba-classes
- https://exceljet.net

Apprendere dal Microsoft Office Dev Center

Il Microsoft Office Dev Center è dedicato ad aiutare i nuovi sviluppatori a imparare a programmare coi prodotti Microsoft Office. Potete accedere alla parte relativa a Excel di questo sito andando su: https://msdn.microsoft.com/en-us/library/office/fp179694.aspx.

Sebbene il sito possa essere di difficile navigazione (oltre a essere solo ed esclusivamente in inglese, almeno al momento), vale la pena di visitarlo per accedere a tutte le sue risorse gratuite, che comprendono codice di esempio, strumenti, istruzioni dettagliate e molto altro ancora.

Sezionare i file di Excel disponibili in azienda

Potete anche trovare l'oro scavando in giardino: i file esistenti in azienda, spesso sono utilissimi per l'apprendimento. Provate ad aprire i file Excel contenenti macro e dategli un'occhiata. Osservate come gli altri membri della vostra azienda utilizzano le macro. Provate ad analizzare le macro riga per riga, per vedere se riuscite a individuare nuove tecniche.

Potreste scoprire alcuni nuovi trucchi, ai quali non avreste mai pensato. Potreste perfino imbattervi in interi pezzi di codice che potrete copiare e implementare nelle vostre cartelle di lavoro.

Chiedere al guru locale di Excel

C'è un genio nella vostra azienda, nel dipartimento, nell'organizzazione o nella comunità? Fate immediatamente amicizia. Avrete a disposizione il vostro personale forum di Excel.

La maggior parte degli esperti di Excel ama condividere le proprie conoscenze. Quindi non abbiate paura di rivolgervi al vostro guru di Excel per porre domande o chiedere consigli su come affrontare i problemi con le macro.

Capitolo 24

Dieci cose da fare e da non fare in VBA

IN QUESTO CAPITOLO

- **Cinque cose relative a VBA che dovreste fare**
- **Cinque cose relative a VBA che non dovreste fare**

Se state leggendo questo capitolo finale, probabilmente avete letto la maggior parte del contenuto di questo libro e avete familiarità con Excel VBA. O magari siete semplicemente saltati alla fine del volume incuriositi dal titolo.

Questo capitolo offre alcuni consigli da tenere in considerazione quando iniziate a sviluppare soluzioni in VBA. Seguire queste linee guida non è una panacea per evitare i problemi di programmazione, ma seguirle può aiutarvi a evitare le insidie nelle quali altri sono incappati.

Dichiarate tutte le variabili

Quanto è comodo: basta iniziare a digitare il codice VBA senza il compito noioso di dichiarare ogni singola variabile che desiderate utilizzare. Sebbene Excel vi permetta di usare variabili non dichiarate, questo può mettervi nei guai.

Il primo comandamento della programmazione VBA dovrebbe essere questo:

Dichiarerai ogni variabile.

Se mancate di autodisciplina, inserite nella parte superiore dei moduli l'istruzione Option Explicit. In questo modo, il codice non verrà nemmeno eseguito se include una o più variabili non dichiarate. Non dichiarare tutte le variabili presenta un solo vantaggio: si risparmia qualche secondo. Ma usando variabili non dichiarate alla fine ne verrete perseguitati. E ci vorrà più di qualche secondo per individuare il problema.

Non confondete password con sicurezza

Avete passato mesi a creare una fantastica app per Excel, con alcune incredibili macro. Siete pronti per divulgarla al mondo, ma non volete che gli altri vedano la vostra incredibile abilità di programmazione delle macro. Basta proteggere con una password il progetto VBA e siete al sicuro, giusto? Sbagliato.

L'utilizzo di una password può impedire agli utenti occasionali di visualizzare il codice. Ma se qualcuno intende *davvero* controllarlo, scoprirà come decifrare la password.

Morale? Se avete assolutamente bisogno di mantenere segreto il vostro codice, Excel non è la scelta migliore come piattaforma di sviluppo.

Pulite il codice

Quando la vostra app funzionerà in modo soddisfacente, dovreste ripulirla. Le attività di pulizia del codice includono quanto segue.

- Assicuratevi che ogni variabile sia dichiarata.
- Assicuratevi che tutte le righe siano indentate correttamente, in modo che la struttura del codice sia evidente.
- Rimuovete eventuali residui del debug, come le istruzioni MsgBox delle istruzioni Debug.Print.
- Rinominate le variabili con nome errato. Per esempio, se utilizzate una variabile UnaVariabile, ci sono buone

probabilità che possiate scegliere un nome più descrittivo. Vi ringrazierete, in seguito.

» I vostri moduli contengono probabilmente alcune routine di “test”, scritte mentre cercavate di capire qualcosa. Hanno raggiunto il loro scopo, quindi potete eliminarle.

» Aggiungete i commenti necessari per capire il funzionamento del codice, utili quando poi lo rivedrete tra sei mesi.

» Assicuratevi che tutto sia scritto correttamente, in particolare il testo nelle UserForm e nelle finestre di messaggi.

» Controllate il codice ridondante. Se disponete di due o più routine con blocchi di codice identici, prendete in considerazione la creazione di una nuova routine richiamabile dalle altre.

Non mettete tutto in una sola routine

Volete creare un programma incomprensibile? Un modo efficace per farlo è quello di mettere tutto il codice all’interno di una sola grande routine. Se doveste utilizzare nuovamente questo programma per apportarvi modifiche, siete destinati a commettere errori e introdurre alcuni interessanti bug, di quelli grossi.

Capite il problema? La soluzione è adottare codice modulare. Suddividete il programma in blocchi più piccoli, ognuno progettato per eseguire un’attività ben specifica. Dopo aver appreso questa abitudine, scoprirete che scrivere codice senza bug è più facile che mai.

Prendete in considerazione altri software

Excel è un programma incredibilmente versatile, ma non è adatto a tutto. Quando siete pronti a intraprendere un nuovo progetto, prendetevi del tempo per considerare tutte le vostre esigenze. Per parafrasare un vecchio detto, “Se tutto ciò che conoscete è il VBA di Excel, tutto vi sembrerà una macro VBA”.

Non date per scontata l’abilitazione delle macro

Come sapete, Excel vi consente di aprire una cartella di lavoro senza però attivare le sue macro. In effetti, è quasi come se i progettisti delle versioni recenti di Excel *volessero* che gli utenti disabilitassero le macro.

Abilitare le macro all’apertura di una cartella di lavoro di provenienza sconosciuta non è mai una buona idea. Questo vi costringe a conoscere i vostri utenti. In alcuni ambienti aziendali, tutte le macro di Microsoft Office sono disattivate e l’utente non ha scelta.

Una cosa da considerare è l’aggiunta di una firma digitale alle cartelle di lavoro che distribuite ad altri. In questo modo, l’utente può essere certo che le cartelle di lavoro vengano effettivamente da voi e che non siano state manipolate. Consultate la Guida per ulteriori informazioni sulle firme digitali.

Prendete l’abitudine di sperimentare

Quando lavorate a un progetto Excel su larga scala, dedicate del tempo a scrivere “esperimenti” VBA di piccole dimensioni. Per esempio, se state cercando di scoprire il funzionamento di un nuovo oggetto, metodo o proprietà, scrivete una semplice routine Sub e giocateci fino a quando non sarete soddisfatti e avrete una conoscenza approfondita di come funziona e dei suoi potenziali problemi.

La semplice sperimentazione è quasi sempre molto più efficiente rispetto all’integrazione di una nuova idea nel codice esistente prima di capire a fondo quale potrebbe essere il risultato.

Non supponete che il vostro codice funzioni con altre versioni di Excel

Quando create un’app Excel, non avete assolutamente alcuna garanzia che essa funzionerà perfettamente nelle versioni precedenti (o future) di Excel. In alcuni casi, le incompatibilità saranno evidenti. Per esempio, se il vostro codice fa riferimento a una nuova funzionalità introdotta in Excel 2019, sapete che non funzionerà nelle versioni precedenti. Ma troverete anche che ci sono cose che dovrebbero funzionare con una versione precedente, ma che non funzionano.

Excel include un pratico controllo di compatibilità (selezionate il comando File ➪ Informazioni ➪ Verifica documento ➪ Verifica compatibilità), il quale però controlla solo la cartella di lavoro e ignora il codice VBA. L’unico modo per essere sicuri che la vostra applicazione funzioni con versioni diverse da quella con cui l’avete creata è provarla fisicamente in quelle versioni.

Fate attenzione ai vostri utenti

Le app di Excel si suddividono in due categorie: quelle che sviluppate per voi stessi e quelle che sviluppate per altri. Se sviluppate app per altri, il vostro lavoro è molto più difficile, perché dovete porre in atto tutta un'altra serie di ipotesi.

Per esempio, potete essere più rilassati nella gestione degli errori, se siete l'unico utente. Se si verificherà un errore, avrete un'idea su come risolverlo. Se qualcun altro sta usando la vostra app e appare lo stesso errore, si sentirà in preda al panico. E quando lavorate con una vostra applicazione, di solito potete cavarvela senza istruzioni.

Dovete capire il livello di abilità di coloro che useranno le vostre cartelle di lavoro e cercare di anticipare i problemi che potrebbero avere. Cercate di mettervi nei panni di un utente della vostra applicazione e identificate tutte le aree che possono causare confusione o problemi.

Non dimenticate i backup

Niente è più scoraggiante di un crash del disco rigido senza avere un backup. Se state lavorando a un progetto importante, ponetevi una semplice domanda: "Se il mio computer bruciasse stasera, che cosa perderei?" Se la vostra risposta è più di qualche ora di lavoro, dedicate del tempo a una routine di backup dei dati. Avete già una routine di backup dei dati, vero?

Indice analitico

A

Abs, funzione
Activate, evento
AddChart2, metodo
Address, proprietà
ambiente VBA, personalizzazione
argomenti
array
  dinamici
  multidimensionali
Array, funzione
attivare VBE
attività, automatizzazione di una
automatizzare
automazione di operazioni ripetitive

B

barra degli strumenti
barra dei menu
Barra indicatori, opzione
barra multifunzione
  aggiungere una macro
  personalizzazione
  personalizzazione con il linguaggio XML
  personalizzazione manuale
BeforeClose, evento
BeforeDoubleClic, evento
BeforeDoubleClick, evento
BeforePrint, evento
BeforeRightClick, evento
BeforeRightClick, evento
BeforeSave, evento
blocchi di commenti, inserimento
blog di esperti
breakpoint, impostazione
bubble-sort, algoritmo
bug
  identificazione di
  suggerimenti per la loro riduzione
  tecniche di eliminazione
  tipi di

C

cartella, ottenere un nome
Casella degli strumenti, controlli disponibili
celle
  copia di un intervallo
  di dimensioni variabili
  riferimenti assoluti
Cells, proprietà
ChangeCase, macro
Change, evento
ChartObject, oggetto
Charts
CheckBox (Casella di controllo), controllo
CheckUser, routine
Choose, funzione
Chr, funzione
cicli
  interni ed esterni
Clear, metodo
codice
  arresto in un punto definito
  cicli nel
  copia del
  copia di più righe contemporaneamente
  esaminare il
  esecuzione passo-passo
  impostare i breakpoint
  inserimento diretto
  passare a una determinata riga

pulizia del
uso dei commenti
Codice, finestra
codice VBA
dove inserire
esempi di
collezioni, definizione
colonna, selezione di una
Colori codice, opzione
Column, proprietà
comando personalizzato, creazione di un
ComboBox (Casella combinata), controllo
CommandButton (Pulsante di comando), controllo
commento
componenti aggiuntivi
aggiungere informazioni descrittive
apertura
concetti di base
creazione
definizione
distribuzione
esempio
impostare la cartella di lavoro
modifica
nuovi per Excel
perché crearli
proteggere il codice VBA
provare la cartella di lavoro
uso dei
concatenare, termine
Const, istruzione
Controllo automatico sintassi, opzione
convalida dei dati
prevenzione dell’eliminazione
Copy, metodo
costanti
predefinite
Count, proprietà
CurDir, funzione
Custom UI Editor

D

data e ora del sistema, visualizzazione
Data, funzione
DateAdd, funzione
DateDiff, funzione
Date, funzione
DatePart, funzione
DateSerial, funzione
DateValue, funzione
dati, convalida dei
Day, funzione
Deactivate, evento
debug
debugger, uso
debugging
tecniche di
Debug.Print, istruzioni
Delete, metodo
Descrizione dati automatica, opzione
dichiarare i tipi di variabile
dichiarazione del tipo di una variabile
Dichiarazione di variabili obbligatoria, opzione
Dimensione, opzione
Dim, istruzione
Dir, funzione
disattivare il calcolo automatico
disattivare l’aggiornamento dello schermo
Do-Until, ciclo
Do-While, ciclo

E

Elenco automatico di proprietà e metodi
Elenco membri automatico
eliminare i fastidiosi messaggi di allarme
EntireColumn, proprietà
Err, funzione
Error, funzione
errori
breve gestione degli
dettagli di gestione
di sintassi

gestire in altro modo gli
intenzionali
recupero
runtime
tecniche di gestione
tipi di
espressioni
di controllo
Espressioni di controllo, finestra
etichette
eventi
attivare e disattivare in una cartella di lavoro
attivare e disattivare in un foglio
attivazione della cartella di lavoro
esempi introduttivi
non associati agli oggetti
per pressioni di tasti
programmare routine di gestione
relativi ai fogli di lavoro
scrivere una routine di gestione
utilità degli
Excel, compatibilità
Exp, funzione
ExtractElement, funzione

F

file, determinare le dimensioni
FileLen, funzione
finestra del codice
finestra Immediata
finestre, riduzione a icona e massimizzazione
finestre di dialogo
aggiungere controlli
checklist
controlli
controlli della
creazione
estetica
gestione dei controlli
non modale a schede
proprietà dei controlli
scrivere il codice
uso delle
visualizzazione
Fix, funzione
foglio di lavoro
font a larghezza fissa
Font, proprietà
For Each-Next, ciclo, con le collezioni
Format, funzione
Formula, proprietà
For-Next, cicli
forum degli utenti
Frame (Cornice), controllo
Function, routine
esecuzione
richiamare da una formula
funzione personalizzata
funzioni
argomenti delle
a singolo elemento
che restituiscono una matrice
con un argomento intervallo
con un argomento opzionale
definizione di
del foglio di lavoro, esempi
di ricerca
esempi VBA
inserimento nel foglio di lavoro
nuove sul foglio di lavoro
personalizzate
personalizzate, debug
senza argomenti
teletrasporto
teletrasporto delle
VBA
visualizzare la descrizione
funzioni del foglio di lavoro
informazioni di utilizzo
funzioni personalizzate
funzioni VBA

G

gerarchia degli oggetti
gestori di eventi
GetOpenFilename, metodo
  argomenti
  esempio d'uso
  sintassi del metodo
GetPrice, routine
GetSaveAsFilename, metodo
  argomenti
GetSetting, funzione
GoTo, istruzione
grafici
  applicazione delle formattazioni
  incorporati
  modifica delle proprietà
  modifica del tipo di
GuessName, macro
Guida di VBA

H

HasFormula, proprietà
Hour, funzione

I

If-Then-Else, struttura
If-Then, struttura
Image (Immagine), controllo
impostazioni booleane, modifica
impostazioni di Excel, modifica delle
impostazioni non booleane, modifica
indicatore di avanzamento, della finestra di dialogo
Informazioni rapide automatiche, opzione
InputBox
  argomenti della funzione
  esempio di
  sintassi
InputBox, funzione
input da tastiera
Inserisci funzione, finestra di dialogo
InStr, funzione
InStrRev, funzione
IntelliSense
Intercettazione degli errori, opzione
interfaccia utente grafica (GUI)
Interior, proprietà
Interrompi ad ogni errore non gestito, opzione
intervallo
  selezione di un
  spostamento di un
Int, funzione
IsArray, funzione
IsDate, funzione
IsEmpty, funzione
IsError, funzione
IsLike, funzione
IsMissing, funzione
IsNull, funzione
IsNumeric, funzione
istruzioni di assegnamento
  esempi
  segno di uguale

L

Label (Etichetta), controllo
LBound, funzione
LCase, funzione
Left, funzione
Len, funzione
ListBox, esempio di
ListBox (Casella di riepilogo), controllo

M

macro
  concetti di base sulla registrazione
  creazione
  creazione della prima
  dare un nome alla

descrizione della
eccesso del registratore
esaminare la
esecuzione utilizzando un tasto di scelta rapida
modificare la
opzioni di registrazione
preparativi per la scrittura
preparazione alla registrazione
registrare in modalità assoluta
registrare in modalità relativa
registrazione della
salvare le cartelle di lavoro
sicurezza delle
test della
uso del registratore
velocizzare le
MAX, funzione
Memorizza macro in, opzione
messaggi d’errore delle istruzioni
metodi
Microsoft Office Dev Center
Mid, funzione
MIN, funzione
Minute, funzione
modalità Interruzione
modello di oggetto
MOD, funzione
moduli
aggiungere un nuovo
creazione
inserire codice VBA in un
rimuovere da un progetto
Month, funzione
MonthName, funzione
MonthNames, funzione
MsgBox
argomenti della funzione
costanti utilizzate
costanti utilizzate per il valore restituito
ottenere una risposta
personalizzazione delle finestre
MsgBox, funzione
MultiPage (Pagine), controllo

N

NewSheet, evento
Now, funzione
NumberFormat, funzione
NumberFormat, proprietà
numero di indice

O

Offset, proprietà
oggetti
Application
concetto di
definizione nella programmazione
esportazione e importazione
eventi per gli
Excel
fare riferimento agli
gerarchia
gerarchia degli
metodi degli
metodi di
modifica delle proprietà
navigare nella gerarchia
proprietà
proprietà degli
semplificare i riferimenti
semplificare i riferimenti agli
specificare un metodo
On Error, istruzione
informazioni
non funziona
OnTime, evento
OnTime, metodo
Open, evento
operatori
OptionButton, utilizzare più insiemi
OptionButton (Pulsante di opzione), controllo
Opzioni, finestra di dialogo

ordine di tabulazione
orologio analogico

P

Paste, metodo
PMT, funzione
procedura
  disfunzionale
  Sub
programmazione
  a oggetti
  orientata agli oggetti
  strutturata
programmi
proprietà
Public, parola riservata
pulsante personalizzato
  creazione di un

R

Range
Range, oggetti
  metodi utili
  proprietà
Range, parola riservata
RefEdit, controllo
Registra macro, finestra di dialogo
Replace, funzione
restituire una lista ordinata
restituire una matrice dei nomi dei mesi
Resume, istruzione
RGB, funzione
Rientro automatico, impostazione
riferimento completo
Right, funzione
rimanere nella routine corrente
riquadro Progetto
Rnd, funzione
routine
  denominazione
  esecuzione
  esecuzione dalla finestra di dialogo Macro
  esecuzione da un pulsante o da una forma
  Function
  Sub
routine di gestione degli eventi
Row, proprietà

S

Salva con nome, finestra di dialogo
SayIt, funzione
scheda Ancoraggio, uso della
scheda Editor, uso della
scheda Formato editor, uso della
scheda Generale, uso della
ScrollBar (Barra di scorrimento), controllo
Second, funzione
Select Case, struttura
  nidificato
SelectionChange, evento
Select, metodo
selezioni multiple
Separatore routine, opzione
SheetActivate, evento
SheetBeforeDoubleClick, evento
SheetBeforeRightClick, evento
SheetChange, evento
SheetDeactivate, evento
Sheets
SheetSelectionChange, evento
Shell, funzione
sistema della Guida
Sorted, funzione
Space, funzione
SpinButton, controllo
SpinButton (Pulsante di selezione), controllo
Split, funzione
spostamento tra moduli e routine
Sqr, funzione
Static, parola riservata

Stop, parola riservata
String, funzione
stringhe
  trovare la lunghezza delle
strutture annidate
Sub, routine
  denominazione
  esecuzione
  esecuzione diretta
  inserimento di un nome
  richiamare la funzione da
Sviluppo, scheda

T

TabStrip (Schede), controllo
tasto di scelta rapida
Tasto di scelta rapida, opzione
terminologia
testo, inserimento di
TextBox
TextBox (Casella di testo), controllo
Text, proprietà
Time, funzione
Timer, funzione
TimeSerial, funzione
TimeValue, funzione
tipi in VBA
Tipo di carattere, opzione
tipo di dati
ToggleButton (Pulsante interruttore), controllo
Trascinamento della selezione, opzione
Trim, funzione
TypeName, funzione

U

UBound, funzione
UCase, funzione
Usa riferimenti relativi, pulsante
UserForm
  aggiungere controlli a una
  aggiungere gli OptionButton
  aggiungere i CommandButton
  alternative
  concetti base, v
  creazione
  esempio
  inserimento di una nuova
  modifica della proprietà di controllo
  quando utilizzare le
  una panoramica della creazione
  uso come indicatore di avanzamento
  uso dei controlli, v
  uso delle
  visualizzare la finestra Codice
  visualizzazione

V

Val, funzione
Value, proprietà
variabili
  cosa sono
  dichiarazione del tipo
  per una sola routine
  per un solo modulo
  pubbliche
  statiche
  vedere per intero il valore
  visibilità
  vita delle
Variabili locali, finestra
VBA, cosa si può fare con
  definizione di
  vantaggi e svantaggi
  velocizzare, suggerimenti
  Visual Basic for Applications
VBE
  attivazione di
  personalizzazione di
video su YouTube
Visual Basic

dieci suggerimenti per l’editor
Visual Basic Editor
Visualizza modulo intero, opzione
Visualizzatore oggetti
VLOOKUP, funzione

W

Weekday, funzione
WindowActivate, evento
WindowDeactivate, evento
With-End With, struttura
Workbooks
Worksheet
Worksheets
wrapper, funzioni, introduzione

Y

Year, funzione

Circa gli autori

Michael Alexander, Microsoft Certified Application Developer (MCAD), è autore di numerosi libri sull'analisi avanzata dei dati con Microsoft Access e Microsoft Excel.

John Walkenbach, soprannominato *Mr Spreadsheet*, è autore di numerosi bestseller su Excel.

Informazioni sul Libro

Lavorare meglio con la programmazione VBA di Excel!

Tutto ciò di cui avete bisogno per iniziare a programmare in VBA. Che desideriate essere più produttivi, automatizzare i compiti più ripetitivi oppure creare le vostre applicazioni, questa guida vi fornisce tutti gli elementi e i concetti essenziali per programmare Excel e per estendere le potenzialità del foglio di calcolo più famoso al mondo con funzioni personalizzate.

https://marapcana.today - sempre online